华为认证系列教程

HCIP-Datacom-Network Automation Developer

Python编程基础

实验指导手册

版本:1.0

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： http://e.huawei.com

华为认证体系介绍

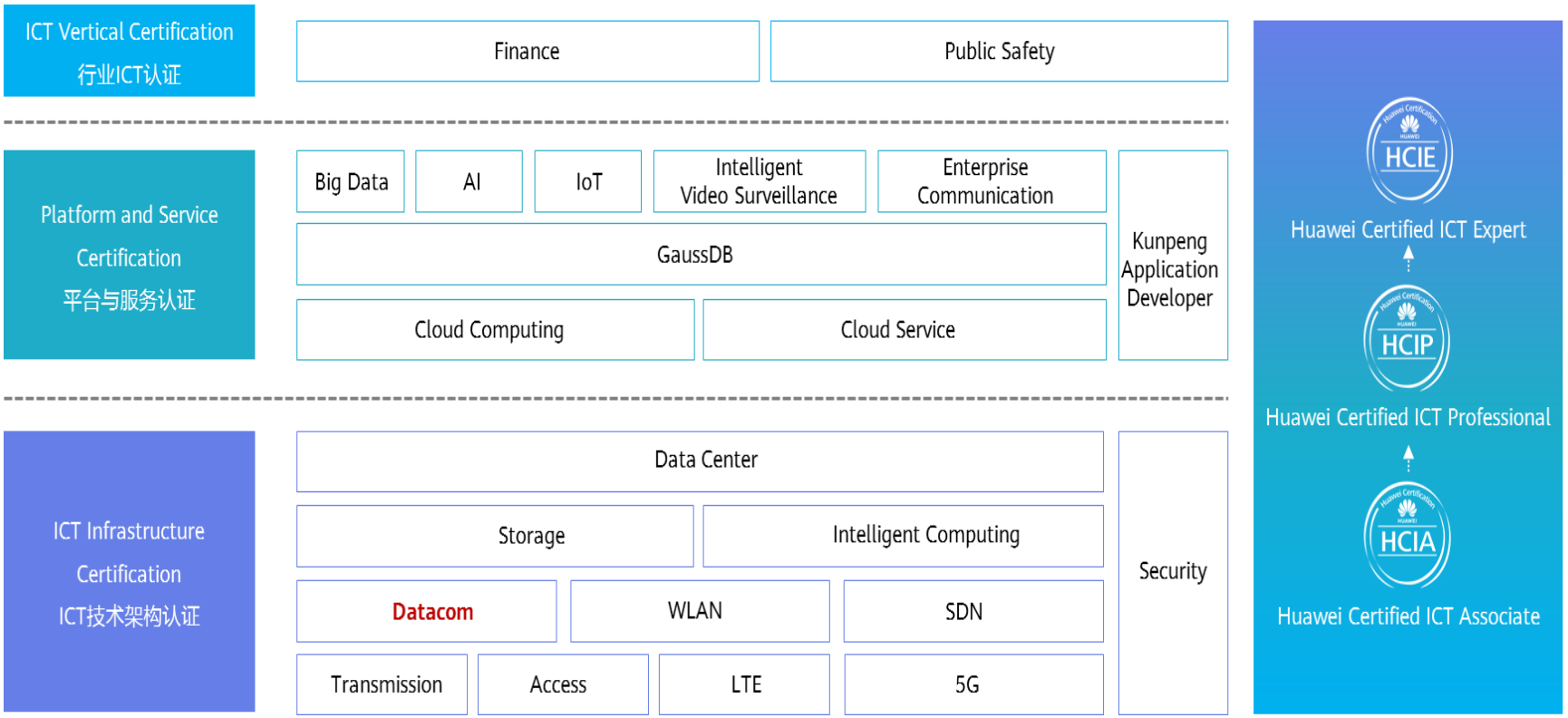
华为认证是华为公司基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，打造的ICT技术架构认证、平台与服务认证、行业ICT认证三类认证，是业界唯一覆盖ICT（Information and Communications Technology 信息通信技术）全技术领域的认证体系。

根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

HCIP-Datacom-Network Automation Developer定位于培养数通网络领域具备网络自动化开发专业知识和技能水平的高级工程师。通过HCIP-Datacom-Network Automation Developer认证将证明您能够胜任企业网络自动化开发工程师岗位，具备使用华为数通设备进行企业网络自动化部署、开发和运维的能力。

华为认证协助您打开行业之窗，开启改变之门，屹立在数通领域的潮头浪尖！

Huawei Certification



前 言

简介

本书为 HCIP-Datacom-Network Automation Developer 认证培训教程，适用于准备参加 HCIP-Datacom-Network Automation Developer 考试的学员，或者希望了解 Python 编程基础知识和实践的读者。

读者知识背景

本文档主要适用于进阶学习的网络自动化工程师。读者需具备以下知识和技能：

- HCIA-Datacom

实验环境说明 ·

环境说明

本实验环境面向准备 HCIP-Datacom-Network Automation Developer 考试的数通网络工程师。本实验可基于任何 Python 编译器，推荐使用 Juypter Notebook 或 Pycharm。

编译环境介绍

本手册开发环境基于编译器版本信息如下：

编译器	软件版本
Anaconda3	2020.02
Pycharm Community Edition	2020.01

目录

1 环境准备

1.1 Anaconda 简介

Anaconda 是一个用于科学计算的 Python 发行版，支持 Linux, Mac, Windows 系统，提供了包管理与环境管理的功能，可以很方便地解决多版本 python 并存、切换以及各种第三方包安装问题。Anaconda 利用工具/命令 conda 来进行 package 和 environment 的管理（也可以使用 pip），并且已经包含了 Python 和相关的配套工具。Anaconda 是适用于企业级大数据分析的 Python 工具。其包含了 720 多个数据科学相关的开源包，在数据可视化、机器学习、深度学习等多方面都有涉及。不仅可以做数据分析，甚至可以用在大数据和人工智能领域。

Anaconda 有如下特性将极大方便开发者的使用：

1. Notebook 对初学者学习友好。

2. Anaconda 已集成 Python，且 python 版本会在下载时进行选择。

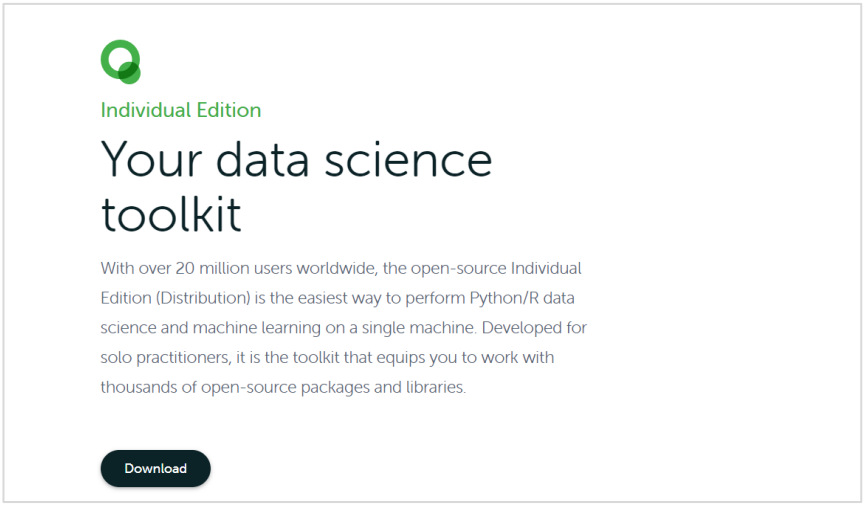
3.开发时，有时需要不同的框架进行支撑，只需要对 anaconda 添加虚拟环境，就可以在不同环境中完成开发，而不需要顾及兼容性等问题，可以为特殊项目配置相应的环境，方便管理。

注：读者也可使用其他开发环境（如 PyCharm）运行 Python 脚本。PyCharm 是一种 Python IDE，带有一整套可以帮助用户在使用 Python 语言开发时提高其效率的工具，比如调试、语法高亮、Project 管理、代码跳转、智能提示、自动完成、单元测试、版本控制。此外，该 IDE 提供了一些高级功能，以用于支持 Django 框架下的专业 Web 开发。

1.2 安装步骤

步骤 1 下载 Anaconda

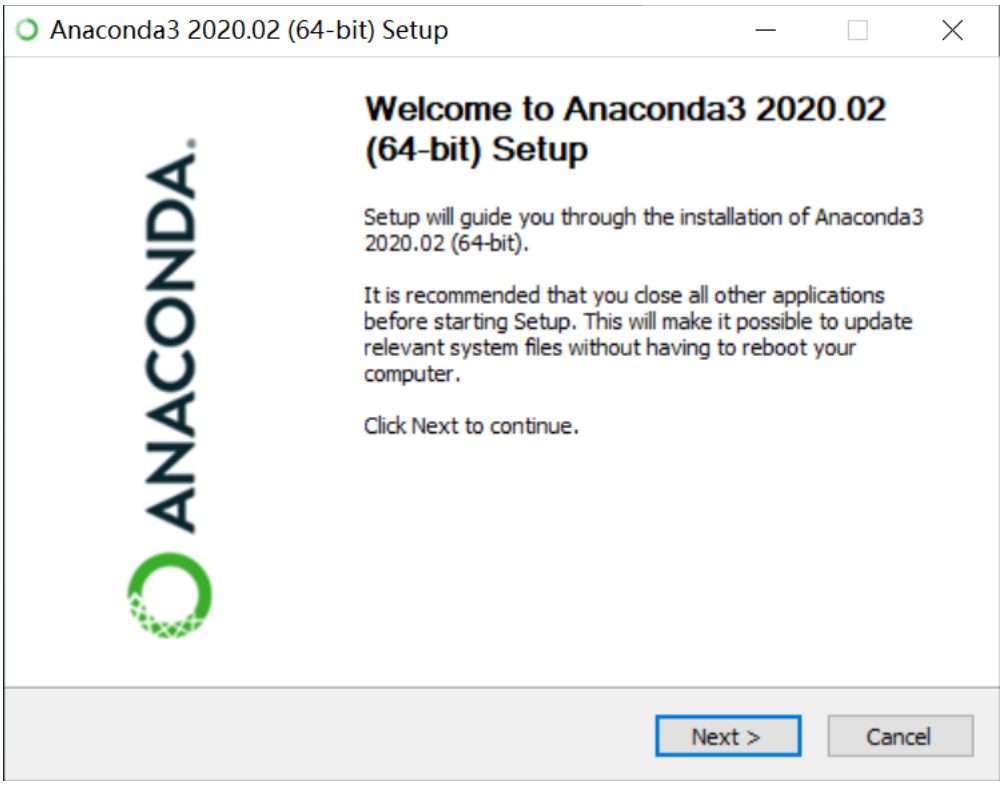
登陆 Anaconda 官网下载安装包 https://www.anaconda.com/products/individual，点击 Download。



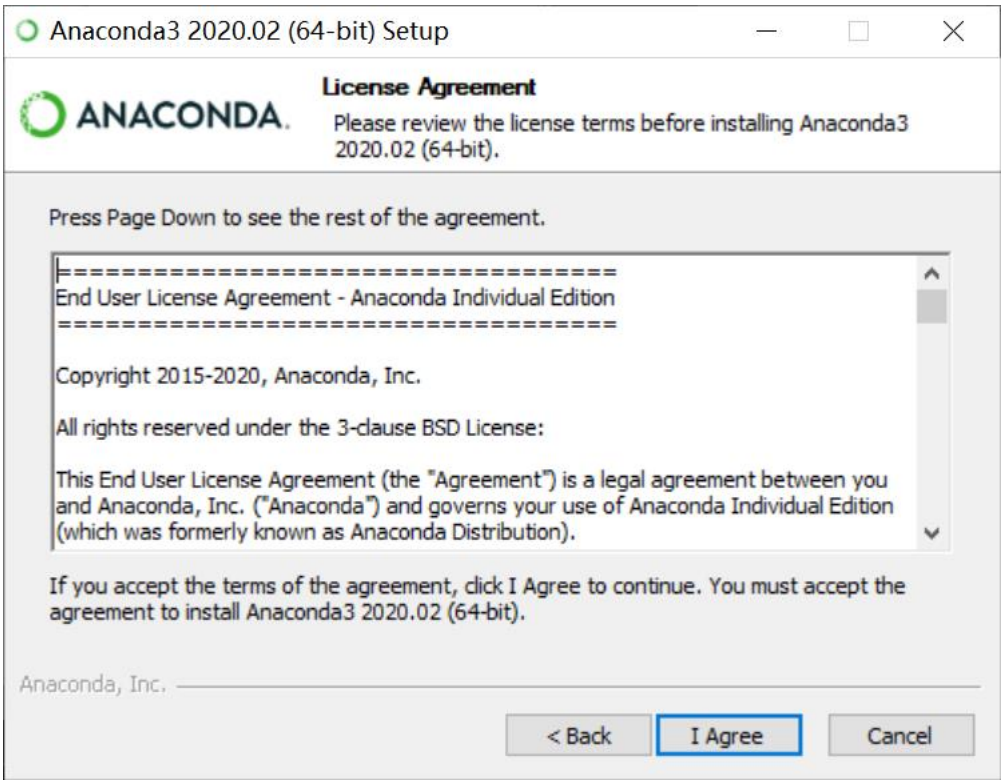
根据系统不同，可以选择 Windows、Mac、Linux 版本，这里选择 Windows，也可以选择 Python3.7 version 或者 Python2.7 version（建议 3.7version），点击 64-Bit Graphical Installer 下载（32 位电脑无法匹配该课程）。

步骤 2 安装 Anaconda

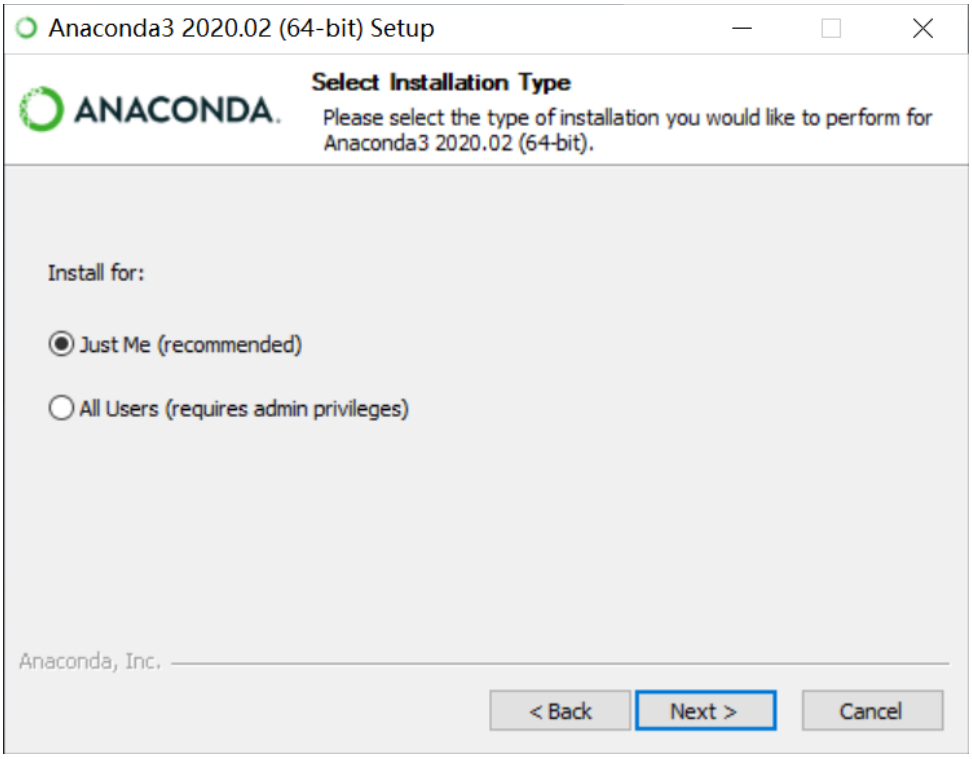
双击下载好的 Anaconda3-x.x.x-Windows-x86_64.exe 文件，出现如下界面，点击 Next。



点击 I Agree。

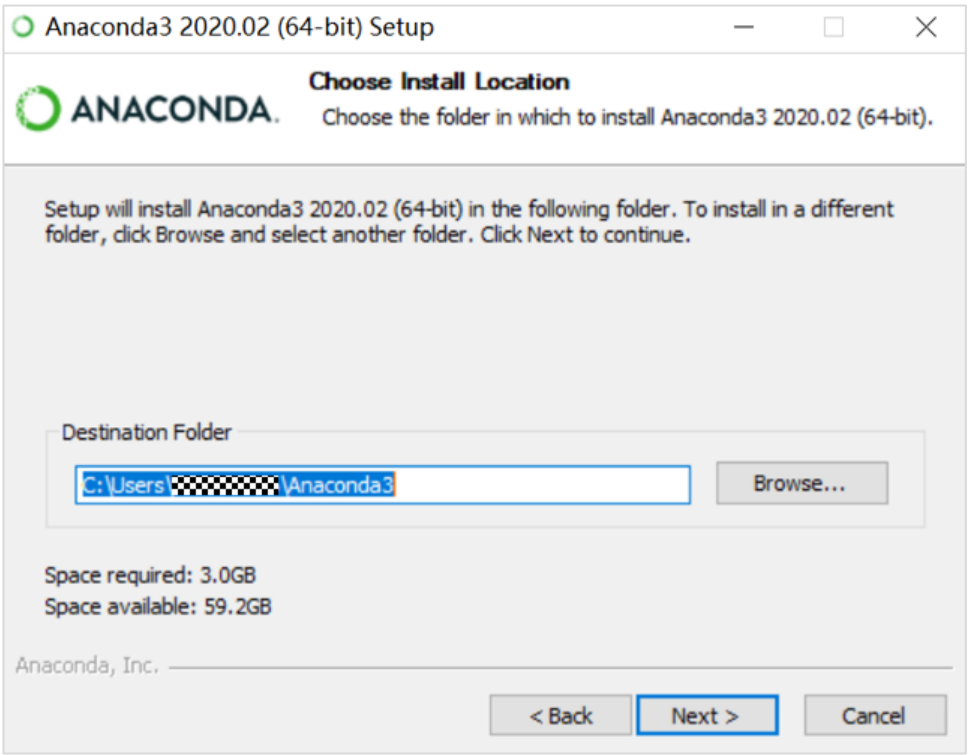


Install for: Just me 还是 All Users,这里直接 Just Me,继续点击 Next。



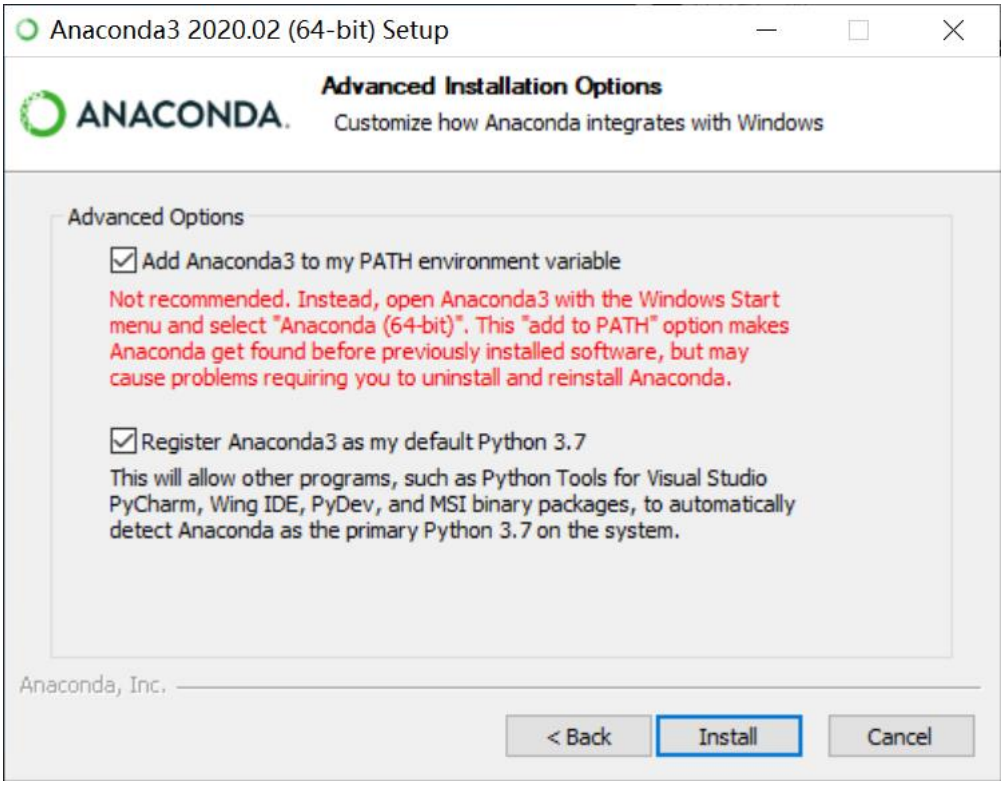
步骤 3 设置安装路径

选择软件安装地址，点击 Next。

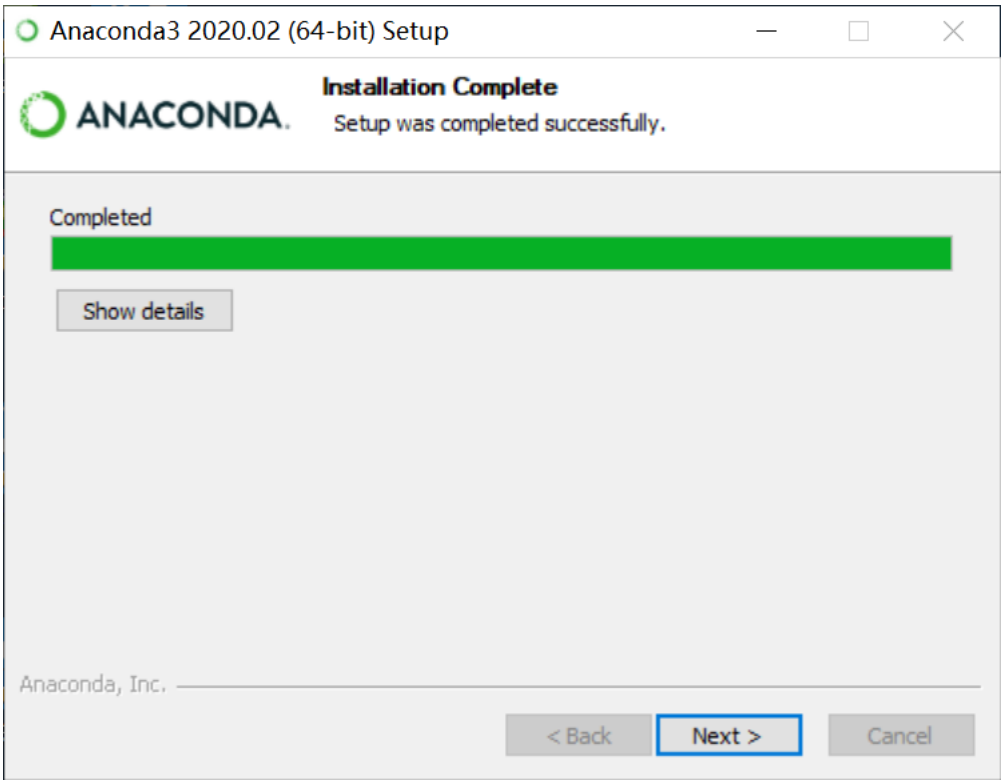


步骤 4 配置环境变量

两个都勾选，第一个是加入环境变量，第二个是默认使用 Python 3.7，可以减少我们后续配置的工作，点击 Install 开始安装。（注意：如果本机器上已经装有 python 的其他版本，建议先删除，然后安装 Anaconda。如未删除，建议两个选项都不勾选，否则容易引发路径错误）

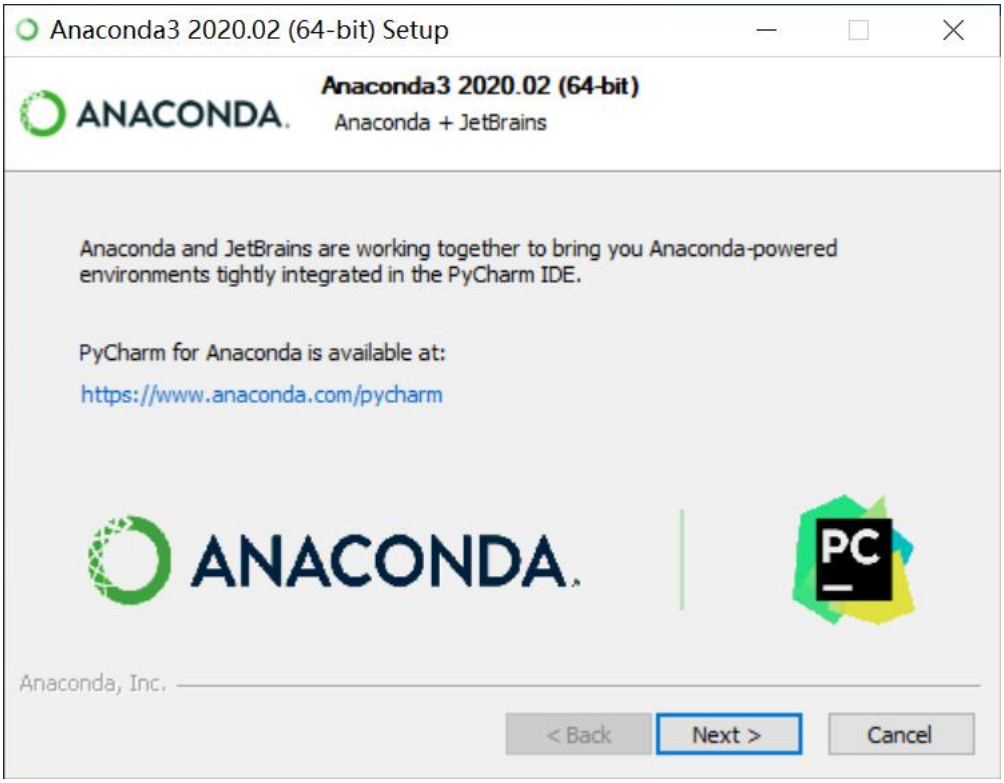


安装完成后点击 Next。

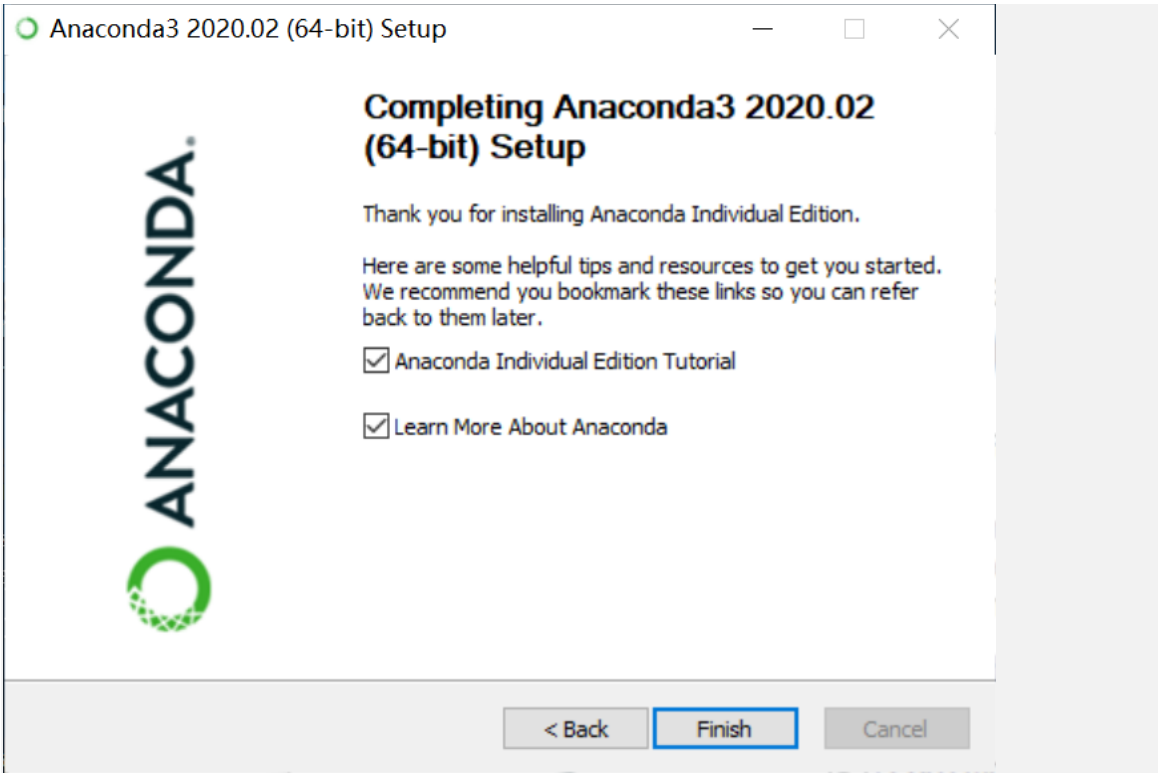


步骤 5 查看安装信息

查看相关信息后点击 Next。



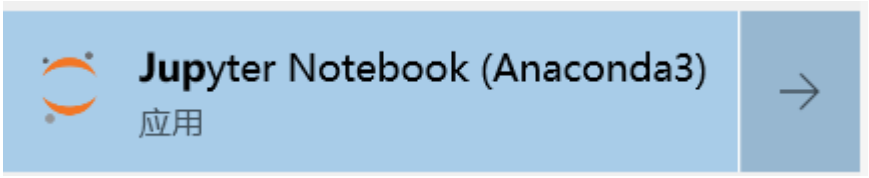
点击 Finish,查看 Anaconda 教程网页。



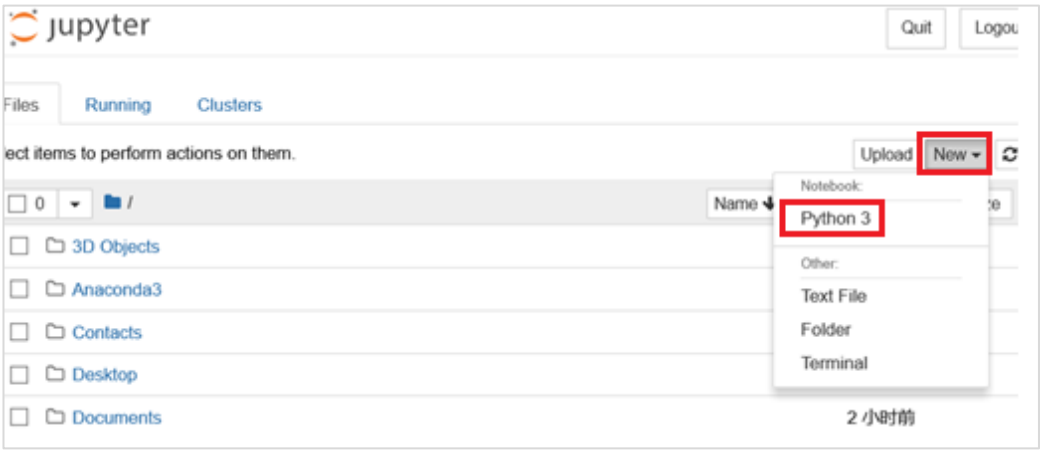
1.3 安装验证

运行 Python 脚本

1.开始菜单找到 Jupyter Notebook，点击并运行，进入 jupyter 主页：

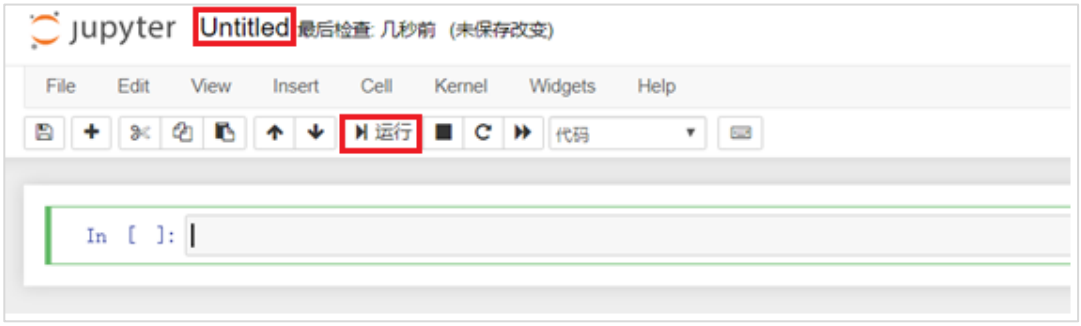


2.点击界面右上方的“New”按钮，选择“Python 3”，创建 jupyter 文件：



上面的红色框为默认的标题，双击可以修改文件名。

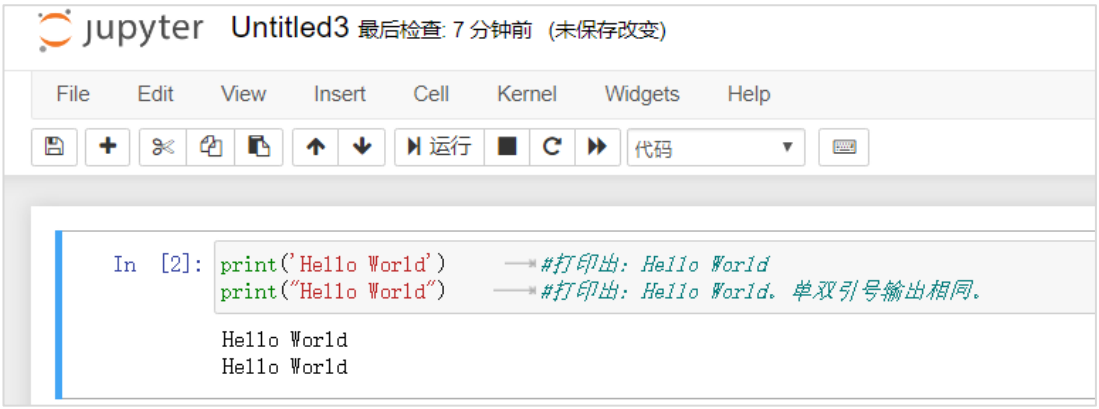
绿色矩形框输入代码。完成代码后点击“运行”运行代码。



3.在输入框输入如下代码：

```
print('Hello World')       #打印出：Hello World
print("Hello World")       #打印出：Hello World。单双引号输出相同。
```

4.点击上方的“运行”按钮，效果如下：



2 Python 编程基础

2.1 实验介绍

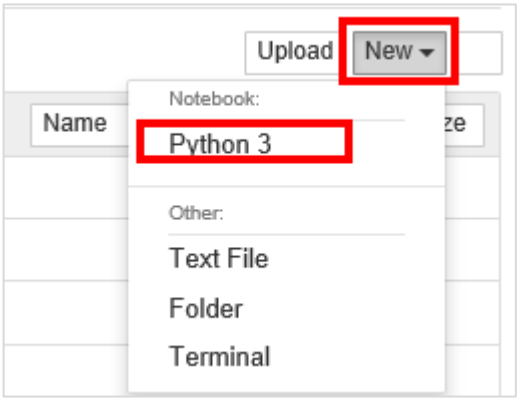
本实验通过各模块独立的代码实践，帮助读者掌握 Python3 语法基础：

- Python 的基本数据类型
- 深拷贝与浅拷贝
- 流控制
- 函数
- 文件操作
- 异常处理
- 面向对象编程

推荐本实验手册结合 Python 官网 Tutorial 学习，https://docs.python.org/3/tutorial/index.html。

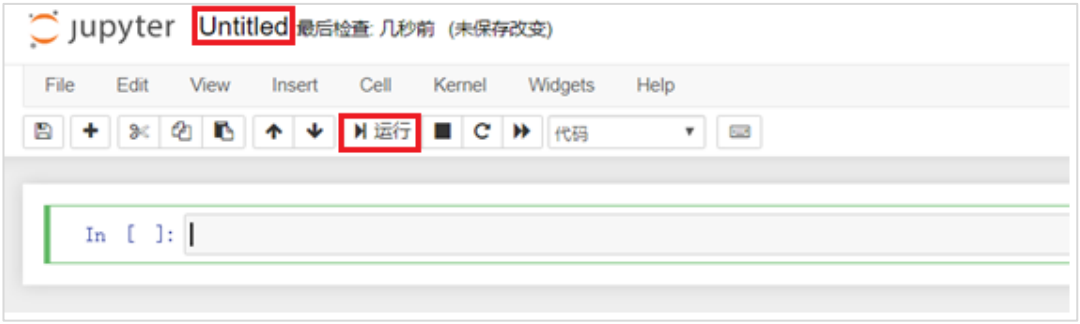
2.2 代码实践

本地完成 Anaconda3 安装后打开 Jupyter Notebook，新建一个 Python3 的 notebook 文件。（读者也可以使用其他编译器实践，例如 Pycharm）



上面的红色框为默认的标题，双击可以修改文件名。

下面的绿色框为代码输入框，完成代码后点击“运行”运行代码。



2.2.1 Python 数据类型

Python 有丰富的数据类型，包括数值、字符串、列表、元组、字典和集合等。本章节通过代码实践理解不同的数据类型的特点。

更多请参考 Python 标准库“内置类型”，https://docs.python.org/zh-cn/3.8/library/index.html。

2.2.1.1 数据类型：数值

数值数据类型用于存储数值。它们是不可改变的数据类型，这意味着改变数值数据类型会分配一个新的对象。

注意，Python 中的“与或非”布尔操作不是使用操作符，而是使用关键词 and/or/not。

```
print(True+False)          # 输出 1，True 默认为 1，False 为 0
print(True or False)       # 输出 True，关键字 or 执行“或”操作
print(5//2)                # 输出 2，//为取整运算符
print(5%2)                 # 输出 1，%为取余运算符
print(3**2)                # 输出 9，**表示乘方操作
print(5+1.6)               # 输出 6.6，不同精度的类型的数字相加默认取高精度类型作为结果
```

2.2.1.2 数据类型：字符串

字符串(String)是由数字、字母、下划线组成的一串字符。由单引号、双引号或者三引号创建对象。

字符串的基本操作：

```
S = 'python'                                # 给变量 S 赋值 python

# len(obj)：返回对象的长度
print(len(S))                               # 输出 6

print(S[0],S[1],S[-1])                      # 输出 pyn ，按照索引获取元素
print(S+'1',S*2)                            # 输出 python1 pythonpython：合并和重复
```

字符串的不可变性：

```
S = 'python'                                # 变量赋值
S[0] = 'Z'                                  # 程序异常，字符串不可变
S1 ='Z'+S[1:]                               # 生成了新的字符串 Zython，并赋值给 S1
                                            # S[1:]代表第一个字符后的字符串，即 ython

print("S:%s,S1:%s"%(S,S1))                  # 输出 S:python，S1:Zython. %s 表示打印字符串
```

字符串的常用操作：

```
S = "python"                                                    # 变量赋值

# str.split(str="", num=-1)：通过指定分隔符对字符串进行切片，如果参数 num 有指定值，则分隔 num+1 个子字符串,-1 表示分割所有。
print(S.split('h'))                                             # 输出[‘pyt’,’on’]，根据 h 对字符串切割

# str.replace(old, new[, max])：返回字符串中的 old（旧字符串） 替换成 new(新字符串)后生成的新字符串，如果指定第三个参数 max，则替换不超过 max 次。
print(S.replace('py','PY'))                                     # PYthon，将字符串中的 py 替换为 PY

# str.upper()：返回小写字符转化为大写后的值。
print(S.upper())                                                # PYTHON

# str.lower()：返回大写字符转化为小写后的值。
print('PYTHON'.lower())                                         # python，字符串转小写

line='aa,bb,ccc,dd\n'                                           # \n 为换行

# str.join(sequence)：sequence：要连接的序列，返回指定字符连接序列中元素后生成的新字符串。
print(''.join(['life ', 'is ' ,'short']))                       # 输出 life is short，join 拼接字符串

hw12='%s %s %d' % ('hello','world',12)                          # 格式化字符串
print(hw12)                                                     # 输出 hello world 12
```

2.2.1.3 数据类型：列表

List（列表）可以完成大多数集合类的数据结构实现。它支持字符，数字，字符串甚至可以包含列表（即嵌套）。它用[ ]标识。

1.列表的常用操作：

```
animals = ['cat', 'dog', 'monkey']                              # 定义列表 animals

# list.append(obj)：在列表末尾添加新的对象。
animals.append('fish')                                          # 追加元素
print(animals)                                                  # 输出 ['cat', 'dog', 'monkey', ‘fish’]

# list.remove(obj)：移除列表中某个值的第一个匹配项。
animals.remove('fish')                                          # 删除元素 fish
print(animals)                                                  # 输出 ['cat', 'dog', 'monkey']

# list.insert(index, obj)：用于将指定对象插入列表的指定位置。index：插入位置
animals.insert(1,'fish')                                        # 在下标 1 的地方插入元素 fish
print(animals)                                                  # 输出 ['cat', ‘fish’, 'dog', 'monkey']

# list.pop([index=-1])：要移除列表中对下标对应的元素（默认是最后一个）。Index：下标
animals.pop(1)                                                  # 删除下标为 1 的元素
print(animals)                                                  # 输出 ['cat', 'dog', 'monkey']
```

2.遍历并获取元素和对应索引

```
#enumerate(sequence) ：将一个可遍历的数据对象组合为一个索引序列，同时列出数据和数据下标，一般用在 for 循环当中。
for i in enumerate(animals):
    print(i)                                    # 元素下标和元素所组成的索引
```

输出：

```
(0, cat)
(1, dog)
(2, monkey)
```

3.列表推导式

```
squares = [x*2 for x in animals]                # 批量生成符合规则的元素组成的列表
print(squares)                                  # ['catcat ', 'dogdog ', 'monkeymonkey ']
```

4.排序

list.sort(cmp=None, key=None, reverse=False)：cmp 为可选参数，如果指定了该参数，会使用该参数的方法进行排序。key 是用来进行比较的元素。reverse 为排序规则，False 为升序。

```
list1 = [12,45,32,55]                           # 定义新列表 list1
list1.sort()                                    # 对列表进行排序
print(list1)                                    # 输出[12,32,45,55]

# list.reverse()：反向列表中元素。
list1.reverse()                                 # 对列表进行逆置
print(list1)                                    # 输出[55,45,32,12]
```

更多字符串方法请参考 https://docs.python.org/zh-cn/3.8/library/stdtypes.html#text-sequence-type-str。

2.2.1.4 数据类型：元组

元组用()标识，内部元素用逗号隔开。元组不能二次赋值，相当于只读的列表。

元组的常用操作：

```
T=(1,2,3)                                       # 创建元组
print(T+(4,5))                                  # 元组合并，输出：(1, 2, 3, 4, 5)
t=(42,)                                         # 只有一个元素的元组，区别于数字
tuple1 = (12,45,32,55,[1,0,3])                  # 创建元祖
tuple1[0] = "good"                              # 程序异常，元组的不可变性，不可赋值
tuple1[4][0] = 2                                # 元组中可变的元素是可以赋值，例子中为[1,0,3]列表中元素
print(tuple1)                                   # (12,45,32,55,[2,0,3])
```

2.2.1.5 数据类型：字典

字典(dictionary)是灵活的内置数据结构类型，字典用"{ }"标识。

字典由索引(key)和它对应的值 value 组成。和列表对比，字典当中的元素是通过键来存取的，而不是通过偏移存取。

字典的常用操作：

```
# 字典的三种赋值操作
x = {'food':'Spam','quantity':4,'color':'pink'}
x = dict(food='Spam',quantity=4, color='pink')
```

```
x = dict([("food", "Spam"),("quantity", "4"),("color","pink")])

# dict.copy()：拷贝数据
d =x.copy()
d['color'] = 'red'
print(x)                                # {'food':'Spam','quantity':4,'color':'pink'}
print(d)                                # {'food':'Spam','quantity':4,'color':'red'}

#元素访问
print(d['name'])                        # 得到错误信息
print(d.get('name'))                    # 输出 None
print(d.get('name','键值不存在！'))      # 输出 键值不存在
print(d.keys())                         # 输出 dict_keys(['food', 'quantity', 'color'])
print(d.values())                       # 输出 dict_values(['Spam', 4, 'red'])
print(d.items())                        # 输出 dict_items([('food', 'Spam'), ('quantity', 4), ('color', 'red')])
d.clear()                               # 清空字典中的所有数据
print(d)                                # 输出 {}
del(d)                                  # 删除字典
```

2.2.1.6 数据类型：集合

集合（set）是一个无序的不重复元素序列。集合可以使用大括号{ } 或者 set() 函数创建。

集合的常用操作：

```
sample_set = {'Prince', 'Techs'}
print('Data' in sample_set)             # 输出 False，in 的作用是检查集合中是否存在某一元素

# set.add(obj)：给集合添加元素，如果添加的元素在集合中已存在，则不执行任何操作
sample_set.add('Data')                  # 向集合中增加元素 Data
print(sample_set)                       # 输出 {'Prince', 'Techs', 'Data'}
print(len(sample_set))                  # 输出 3

# set.remove(obj)：移除集合中的指定元素。
sample_set.remove('Data')               # 删除元素 Data
print(sample_set)                       # {'Prince', 'Techs'}

list2 = [1,3,1,5,3]
print(set(list2))                       # 输出{1, 3, 5}，利用集合元素的唯一性进行列表去重
print(list(set(list2)))                 # 输出 [1,3,5]列表
sample_set = frozenset(sample_set)      # 不可变集合
```

2.2.2 深拷贝与浅拷贝

在 Python 中，对象赋值实际上是对象的引用。当创建一个对象把它赋给另一个变量的时候，Python 并没有拷贝这个对象，而只是拷贝了这个对象的引用。

本实验使用 Python 中的 copy 模块来理解深拷贝和浅拷贝的区别。

```
import copy                             # 当前程序调用 copy 模块
Dict1 = { 'name': 'lee', 'age':89, 'num':[1,2,8]}   # 新建字典
Dict_copy = Dict1.copy()                # 浅拷贝
Dict_dcopy = copy.deepcopy(Dict1)       # 深拷贝
```

```
Dict2=Dict1                                    # 浅拷贝,直接赋值对象
Dict1['num'][1] = 6                            # 修改原数据中嵌套列表的值
print('Dict1:'+str(Dict1)+"\n",'Dict_copy:'+ str(Dict_copy)+"\n",'Dict_dcopy:'+ str(Dict_dcopy)+
"\n",'Dict2:'+str(Dict2))
```

输出结果：

```
Dict1:{‘name’:’lee’, ‘age’:89, ‘num’:[1,6,8]}
Dict_copy :{‘name’:’lee’, ‘age’:89, ‘num’:[1,6,8]}        # 浅拷贝数据一起被修改
Dict_dcopy :{‘name’:’lee’, ‘age’:89, ‘num’:[1,2,8]}       # 深拷贝数据没有修改
Dict2:{'name': 'lee', 'age': 89, 'num': [1, 6, 8]}        # 对象赋值为浅拷贝，数据被修改
```

2.2.3 if 语句

编程语言提供了各种控制结构，允许更复杂的执行路径。if 语句就是其中的一种，用于匹配条件后执行后续指令。

本例编写 Python 脚本，作用是接收一个用户输入的分数，然后判断用户所输入的分数属于什么级别。使用 Python 中的 if 语句可以完成此功能。

```
#根据输入的分数判断
# input()：用于接收输入。
score = input("请输入你的分数")                          # input 函数接收输入，为字符串类型
# try:… except Exception:… 是 Python 中用于捕获异常的语句，如果 try 中的语句出现错误，则会执行 except
中的语句。
try:
    score = float(score)                                 # 将分数转化为数字类型
    if 100>=score>=90:                                   # 判断输入的值是否大于等级分数
        print("优")                                      # 满足条件后输出等级
    elif 90 > score >= 80:
        print("良")
    elif 80>score>=60:
        print("中")
    elif 60>score>=0:
        print("差")
    else:
        raise
except Exception:
  print("请输入正确的分数")
```

2.2.4 循环语句

循环语句允许多次执行一个语句或语句组。本例介绍两种循环，for 和 while。

1. for 循环

Python for 循环可以遍历任何序列的项目，如一个列表或者一个字符串。

本例中编写 Python 脚本，打印九九乘法表。

```
for i in range(1,10):                                    # 定义外层循环
    for j in range(1,i+1):                               # 定义内层循环
        print ("%d * %d = %2d" % (i , j , i*j), end=" \t ")  # 字符串的格式化输出，让打印结果进行对齐
    print()                                              # end 属性设置打印结尾符号默认为\n
```

输出结果：

```
1*1= 1
2*1= 2 2*2= 4
3*1= 3 3*2= 6 3*3= 9
4*1= 4 4*2= 8 4*3=12 4*4=16
5*1= 5 5*2=10 5*3=15 5*4=20 5*5=25
6*1= 6 6*2=12 6*3=18 6*4=24 6*5=30 6*6=36
7*1= 7 7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49
8*1= 8 8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64
9*1= 9 9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81
```

2.while 循环

Python 编程中 while 语句用于循环执行程序，即在某条件下，循环执行某段程序，以处理需要重复处理的相同任务。

当满足条件时循环执行语句块，想要结束循环时，使用 break 或 continue 结束循环。

```
#while 循环
i = 0                                          # 新建 i 变量
while i<9:                                     # 设置循环条件
    i+=1                                       # 每次循环 i 增加 1
    if i == 3:                                 # 判断条件是否满足
        print("跳出此次循环")
        continue                               # continue 跳出当前的这一次循环
    if i == 5:
        print("跳出当前大的循环")
        break                                  # 跳出当前的大的循环
    print(i)
```

输出结果：

```
1
2
跳出此次循环
4
跳出当前大的循环
```

2.2.5 自定义函数

函数是组织好的、可重复使用的一段代码。它能够提高程序的模块化程度和代码利用率。Python 提供很多内建的函数，例如前面使用过的 print()。还可以自己创建函数，也就是自定义函数。

本小结将介绍自定义函数。

1. 定义函数

使用 def 关键字自定义一个函数，能够调用函数功能（打印信息）。

```
def Print_Hello ( ):                    # 函数名称，无输入值
    print ('Hello, Python.')           # 输出
Print_Hello()                           # 函数调用
```

输出结果：

```
Hello, Python.
```

2. 函数的参数传递与默认参数

自定义函数，要求打印不同方式传入的参数。

```
def hello(greeting='hello',name='world'):          # 默认参数
    print('%s, %s!' % (greeting, name))            # 格式化输出
hello()                                            # hello，world      无参数则使用默认参数
hello('Greetings')                                 # Greetings，world       位置参数
hello(name='Huawei')                               # hello，Huawei关键字参数
hello('Greetings','universe')                      # Greetings，universe       位置参数
```

输出结果：

```
hello, world!
Greetings, world!
hello, Huawei!
Greetings, universe!
```

3. 不定长参数

不定长参数为希望函数处理的参数个数比在定义的时候多。此时我们使用* 表示元组类型参数，**表示字典类型的参数。

```
def test (a, b, *c, **d):
    print (a)
    print (b)
    print (c)
    print (d)
test (1,2,3,4,5,6,x=10,y=11)          # 传递的参数会自动区分元组或字典类型
```

输出结果：

```
1
2
(3, 4, 5, 6)
{'x': 10, 'y': 11}
```

4. 返回值

所谓的函数返回值，是函数在完成方法后给调用者的结果。使用 return 语句来完成。

```
def plus_one (number):
    return int(number)+1
plus_one (10)
```

输出结果：

```
11
```

2.2.6 IO 操作

Python 标准库非常庞大，提供组件的功能范围十分广泛。文件 IO 就是其中一个内置的模块。更多的标准库操作请参考官方文档 https://docs.python.org/zh-cn/3.8/library/index.html。

Python IO 操作为对文件的读写，通常我们可以使用 open 函数完成操作。

文件模式	说明
r	默认模式，只读模式。文件指针会放在文件头。
w	只写入模式。文件指针会放在文件头，文件覆盖原内容。
a	追加模式，指针在文件结尾。若无文件则创建新文件。
rb	以二进制格式只读文件。（例如声音或图像文件）
wb	以二进制格式只写文件。
ab	以二进制格式追加文件。
r+	可读可写，文件指针会放在文件头。
w+	可读可写。如果文件存在将覆盖文件，如果不存在则创建新文件。
a+	追加可读可写。若无文件则创建新文件。

1. 文件写入

```
f = open("text.txt", 'w')                    # 打开文件 text.txt，当文件不存在时会新建一个。w 表示文件模式为写入
Str = input("请输入要写入的内容：")          # 内置函数 input 获取控制台的输入
f.write(Str)                                 # 将 Str 写入到文件 f
f.close()                                    # 关闭文件
```

输出结果：

输入写入内容“python 文件操作”。

在同一目录生成此 text.txt 文件。

2. 文件读取

文件读取使用 read 方法，read()如果不指定大小表示全部读取。

```
f = open("text.txt", 'r')                    # r 表示文件描述为读取
print(f.read(6))                             # 读取六个字符，当前光标后移六个字符
print(f.read())                              # 读取光标所在位置至最后
f.close()
```

输出结果：

```
python
文件操作
```

3. 使用上下文管理器操作文件

上下文管理器规定了资源的使用范围。代码开始时候分配资源，结束时释放资源。能够很好的精简代码和提高代码的可读性。上下文管理器和 with 语句一起工作。基本语法为：

with 上下文表达式 [as 资源对象]：#as 表示将上下文的返回值赋值给资源对象

对象的操作

```
with open("text1.txt", 'w') as f:          # 使用 with 语句进行文件写入
    f.write("python 文件操作")
with open("text1.txt", 'r') as f:          # 使用 with 语句读取文件内容
    print(f.read())
```

输出结果：

```
python 文件操作
```

文件操作有更多的方法，例如读取一行 readlines 方法，文件定位 tell 方法，seek 方法等。可参考 Python Tutorial 进行进一步学习。

2.2.7 异常处理

程序在执行过程中产生的错误称为异常。常见的异常有语法错误（SyntaxError），未声明变量（NameError），访问未知属性（AttributeError）等。Python 有强大的异常处理机制，它可以准确的反馈错误信息，帮助开发人员定位。

Python 使用 try-except 语句处理异常。其中 try 语句用于检查异常，except 用于捕获异常。

1. 制造异常

首先我们来制造一个异常，除法除数为 0。

```
num1 = input ('请输入第一个数字：')
num2 = input ('请输入第二个数字：')
print ('第一个数字除以第二个数字为：%f'%(int(num1)/int(num2)))    # %f 代表浮点数
```

输出结果：

```
请输入第一个数字：10
请输入第二个数字：0
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-10-5cc9c0d19753> in <module>
      1 num1 = input ('请输入第一个数字：')
      2 num2 = input ('请输入第二个数字：')
----> 3 print ('第一个数字除以第二个数字为：%f'%(int(num1)/int(num2)))
      4
ZeroDivisionError: division by zero
```

我们可以看到异常原因为 ZeroDivisionError。我们可以捕获这个异常，保持程序正常运行。

2. 使用 try-except 生成代码

使用 try-except 结构捕获到 ZeroDivisionError 这个异常，输出“第二个数不能为 0”。

```
try:
    num1 = input ('请输入第一个数字：')
    num2 = input ('请输入第二个数字：')
    print ('第一个数字除以第二个数字为：%f'%(int(num1)/int(num2)))
except ZeroDivisionError:
    print ('第二个数不能为 0')
```

输出结果：

```
请输入第一个数字：10
请输入第二个数字：0
第二个数不能为 0
```

成功捕获此异常。如果需要捕获多个异常，可以在并列有多个 except 语句捕获不同异常。

3. 捕获所有异常

如果每个异常都需要定义非常麻烦。Exception 类是所有异常的父类，可用来表示所有异常。

```
try:
    num1 = input ('请输入第一个数字：')
    num2 = input ('请输入第二个数字：')
    print ('第一个数字除以第二个数字为：%f'%(int(num1)/int(num2)))
except Exception as result :
    print ('捕获到异常:%s' % result)
```

输出结果：

```
请输入第一个数字：hello
请输入第二个数字：0
捕获到异常:invalid literal for int() with base 10: 'hello'
```

4. 没有捕获到异常（else 和 finally）

在程序中如果没有捕获到异常，程序会执行 else 语句。还有情况是无论是否捕获到异常，都要执行一些终止行为这时我们可以用 finally 语句。

Python 中完整的异常处理格式为：

```
try:
    #语句
except:
    #异常处理代码
except:
    #另一个异常处理代码
else:
    #没有异常的执行代码
finally:
    #最后必须执行的代码，无论是否有异常
```

2.2.8 面向对象编程（类）

类和对象是面向对象编程的重要概念。对象是具体的事物，类是对一群具有相同特征和行为的事物的统称，中文里我们可以理解为“分类”。

例如第一个案例中，狗是一个类，而哈士奇是其中的一个对象，哈士奇具备狗这个类的所有特征和行为。

2.2.8.1 创建和使用类

创建 Dog 类。

根据 Dog 类创建的每个实例都将存储名字和年龄。我们将赋予了每条小狗蹲下 sit（ ）和打滚 roll_over（ ）的能力：

```
class Dog():                                    # 使用 class 关键字声明一个类
    """模拟狗的行为（方法）"""
    def sit(self):                              # 使用函数定义类的方法，self 表示自身，永远为第一个参数
        """模拟小狗被命令时蹲下"""
        print("Dog is now sitting")             # 方法中使用 self 访问了 name 属性
    def roll_over(self):
        """模拟小狗被命令时打滚"""
        print("Dog rolled over!")
dog = Dog()                                     # 创建一个对象，并用 dog 保存它的引用
dog.name = "哈士奇"                             # 添加表示名字的属性，name 为哈士奇
dog.sit()                                       # 调用方法
dog.roll_over()
print (dog.name)
```

输出结果：

```
Dog is now sitting
Dog rolled over!
哈士奇
```

2.2.8.2 类的属性

上一个案例中如果要再创建一个 Dog 类的对象，需要再次使用”对象名.属性名”的形式添加，比较麻烦。为了解决这个问题，Python 提供了一个构造方法__init__（前后两个下划线），实现类的初始化实现。

本例中，我们将 name 和 age 两个属性初始化到 Dog 类中。

```
class Dog():                                    # 使用 class 关键字声明一个类
        #模拟狗的属性和行为（方法）
    def __init__ (self,name,age):               # 初始化类的属性，第一个参数永远都是 self，表示对象自身
        #初始化属性 name 和 age
        self.name = name
        self.age = age
    def sit(self):                              # 使用函数定义类的方法，携带 self 参数
        #模拟小狗被命令时蹲下
        print(self.name+" is now sitting")      # 方法中使用 self 访问了 name 属性
    def roll_over(self):
        #模拟小狗被命令时打滚
        print(self.name+" rolled over!")
dog = Dog("哈士奇",2)                           # 创建一个对象，并用 dog 保存它的引用
print (dog.age)                                 # 访问属性
dog.sit()                                       # 调用方法
dog.roll_over()
```

输出结果：

```
2
哈士奇 is now sitting
哈士奇 rolled over!
```

2.2.8.3 类的封装

我们把隐藏属性、方法与方法实现的细节的过程称为封装。通过将属性定义为私有属性，避免外界随意赋值。

私有属性通过__ (两个下划线前缀)实现。

```
class Dog():
    def __init__ (self,name,age):
        self.name = name
        self.__age = age                    # 将 age 设置为私有属性__age
    def get_age(self):
        return self.__age
dog = Dog('哈士奇','2')
print (dog.name)
dog.get_age()                               # 调用 get_age()方法，返回参数
print (dog.__age)                           # 这里程序将报错，显示没有__age 属性，外部无法直接调用
```

输出结果：

```
哈士奇
2
AttributeError                            Traceback (most recent call last)
<ipython-input-23-374e764e1475> in <module>
      5 dog = Dog('哈士奇','2')
      6 print (dog.name)
----> 7 print (dog.__age)
      8
AttributeError: 'Dog' object has no attribute '__age'
```

2.2.8.4 类的继承

面向对象的编程带来的主要好处之一是代码的重用，实现这种重用的方法之一是通过继承机制。继承完全可以理解成类之间的类型和子类型关系。

本例中构建一个 Dog 的子类 Husky，继承父类的属性和方法。使用方法为 class 子类(父类)。如果一个子类需要多继承，方法为 class 子类（父类 1，父类 2）。本案例为单继承。

```
class Dog():
    def __init__ (self,name,age):
        self.name = name
        self.__age = age            # 将 age 设置为私有属性__age
    def get_age(self):
        return self.__age
class Husky(Dog):                   # 声明一个子类哈士奇，父类为 Dog
    pass
mydog = Husky('Larry',3)            # 子类继承了父类的构造方法
print (mydog.name)                  # 子类继承了父类的 name 属性，私有属性无法继承
mydog.get_age()                     # 子类继承了父类的 get_age()方法
```

输出结果：

```
Larry
3
```

2.2.8.5 类多态

在 Python 中，多态指在不考虑对象类型的情况下使用对象。意思是 Python 不关心对象是父类还是子类，只关心对象的行为（方法）。

本例中父类为 Dog，有 shout 方法。Dog 的子类哈士奇和藏獒，重写了父类的 shout 方法。我们定义一个函数 sound，调用 shout 方法可以不用关心对象是父类还是子类，只要他们都有这个 shout 的方法。

```
# 父类 Dog，叫声为 Dog shouts
class Dog():
    def __init__ (self,name,age):
        self.name = name
        self.__age = age
    def get_age(self):
        return self.__age
    def shout(self):
        print ('Dog shouts')

# 子类哈士奇，叫声为呜呜
class Husky(Dog):
    def shout(self):
        print('呜呜')

# 子类藏獒，叫声为汪汪
class Tibetan_Mastiff(Dog):
    def shout(self):
        print('汪汪')

# 定义一个函数 sound，调用 shout 方法
def sound(dog):
    dog.shout()

# dog1 为哈士奇的对象
dog1 = Husky('Larry',3)
sound(dog1)

# dog2 为 Dog 的对象
dog2=Dog('Richard',2)
sound(dog2)

# dog3 为藏獒的对象
dog3= Tibetan_Mastiff('Sylvia',4)
sound(dog3)
```

输出结果：

```
呜呜
Dog shouts
汪汪
```

3 Python 高级

3.1 实验介绍

本实验通过各模块独立的代码实践，帮助读者掌握 Python3 高级语法：

- 正则表达式
- 装饰器
- 迭代器
- 生成器
- 多任务

3.2 代码实践

3.2.1 正则表达式

正则表达式是一个特殊的字符序列，它能帮助你方便的检查一个字符串是否与某种模式匹配。在 Python 中我们使用 re 模块实现正则表达式的功能。

表3-1 基本的正则表达式语法

符号	含义	示例
^	表示匹配字符串的开始位置	
$	表示匹配字符串的结束位置	
*	表示匹配零次到多次	
+	表示匹配一次到多次（至少有一次)	
?	表示匹配零次或一次	
.	表示匹配单个字符	
|	表示为或者,两项中取一项	a|b 匹配a或者b
( )	小括号表示匹配括号中全部字符	(abc) 匹配abc

[ ]	中括号表示匹配括号中一个字符	[0-9 a-z A-Z]
{ }	大括号用于限定匹配次数	{n}表示匹配n个字符 {n,}表示至少匹配n个字符 {n,m}表示至少n,最多m
\	转义字符	* 表示匹配*号
\w	表示匹配英文字母和数字	
\W	匹配非英文字母和数字	
\d	匹配数字	
\D	匹配非数字	

表3-2 Python 正则表达式 flag 标志

修饰符	描述
re.I	使匹配对大小写不敏感
re.M	多行匹配，影响 ^ 和 $
re.S	使 . 匹配包括换行在内的所有字符
re.U	根据Unicode字符集解析字符。这个标志影响 \w, \W

本教程仅简单介绍正则表达式基本用法，更多请参考官方文档。

3.2.1.1 re.match

re.match 尝试从字符串的起始位置匹配一个模式，如果不是起始位置匹配成功的话，match() 就返回 none。

函数语法：

re.match(pattern, string, flags=0)

其中 pattern 表示匹配的正则表达式。string 表示要匹配的对象。flags 为标志位，例如是否区分大小写，具体可以参考 tutorial 查看细节。

示例：

```
import re
print(re.match('www', 'www.huawei.com'))     # 默认在起始位置匹配 www
print(re.match('com', 'www.huawei.com'))     # 无法匹配
print(re.match('.*com$','www.huawei.com'))   # 正则表达式表示匹配末尾为.com 的对象
```

输出结果：

```
<re.Match object; span=(0, 3), match='www'>
None
<re.Match object; span=(0, 14), match='www.huawei.com'>
```

3.2.1.2 re.search

re.search 的作用是扫描整个字符串并返回第一个成功的匹配。

函数语法：

re.search(pattern, string, flags=0)

示例：

```
import re
print(re.search('www', 'www.huawei.com'))          # 搜索到开始 www
print(re.search('com', 'www.huawei.com'))          # 搜到到最末 com
print(re.search('.*com$','www.huawei.com'))        # 正则表达式表示匹配末尾为.com 的对象
print(re.search('COM', 'www.huawei.com',re.I))     # 使大小写不敏感
```

输出结果：

```
<re.Match object; span=(0, 3), match='www'>
<re.Match object; span=(11, 14), match='com'>
<re.Match object; span=(0, 14), match='www.huawei.com'>
<re.Match object; span=(11, 14), match='com'>
```

3.2.1.3 检索和替换

Python 的 re 模块提供了 re.sub 用于替换字符串中的匹配项。

语法：

re.sub(pattern, repl, string, count=0, flags=0)

pattern 表示正则表达式。repl 表示替换成的字符串，也可以为函数。String 表示要被查找的原始对象。Count 为最大替换次数，0 表示所有。

示例：

```
import re
print(re.sub('^WWW','support', 'www.huawei.com',flags=re.I))    # 将开始的 www，替换为 support
```

输出结果：

```
support.huawei.com
```

3.2.1.4 re.compile

compile 函数用于编译正则表达式，生成一个正则表达式（ Pattern ）对象，供 match() 和 search() 这两个函数使用。

语法格式为：

re.compile(pattern[, flags])

示例：

```
import re
pattern = re.compile('\d+')                          # 匹配至少一个数字
print (pattern.match( 'www.huawei123.com'))    # 查看头部是否有数字，没有匹配
print (pattern.search( 'www.huawei123.com'))    # 搜索返回第一个数字
```

输出结果：

```
None
<re.Match object; span=(10, 13), match='123'>
```

3.2.1.5 findall

findall 表示在字符串中找到正则表达式所匹配的所有子串，并返回一个列表，如果没有找到匹配的，则返回空列表。

语法格式为：

findall(string[, pos[, endpos]])

string 为待匹配字符串。Pos 指定字符串的起始位置，默认为 0。Endpos 指定字符串结束位置，默认为字符串长度。

示例：

```
import re
pattern = re.compile('\d+')                                  # 匹配至少一个数字
print(pattern.findall( 'www.huawei123.com \n 345'))    # 查找所有字符串
```

输出结果：

```
['123', '345']
```

3.2.1.6 re.split()

split 按照能够匹配的子串将字符串分割后返回列表，它的使用形式如下：

re.split(pattern, string[, maxsplit=0, flags=0])

pattern 为正则表达式。string 为要匹配的字符串。Maxsplit 为分割次数，默认为 0 不限次数。

示例：

```
import re
s = re.split('\W+', 'www.huawei.com')              # \W+表示匹配非英文和数字一次或多次
print(s)
```

输出结果：

```
['www', 'huawei', 'com']
```

3.2.2 装饰器

装饰器本质是一个 Python 函数，特殊之处在于它的返回值也是一个函数。它可以在不改动现有函数的前提下，对函数的功能进行扩展。装饰器的语法是以@开头，具体用法如下：

1. 构建初始函数

```
现有一个函数，输出 I am learning Decorators.
def test():
    print ("I am learning Decorators.")
test()
```

输出结果：

```
I am learning Decorators.
```

2. 扩展函数功能需求

现在我们想对函数功能进行扩展，在之前需要输出 I learned Python Data Structures.

一般情况下我们的解决办法是。

```
def test():
```

```
    print ("I learned Python Data Structures.")
    print ("I am learning Decorators.")
```

这样可以解决问题，但是改变的现有的函数。如果我们想不改变原函数情况下实现此功能，可以使用装饰器

3. 装饰器装饰函数

定义一个新的函数 decorator()，它的参数是一个函数，返回值也是一个函数。这个作为参数的函数 func()在返回函数 wrapper()内部执行。然后在 test()函数前加上@decorator，就像被新装饰这个新的函数。

```
def decorator(func):
    def wrapper():
        print ("I learned Python Data Structures.")
        func()
    return wrapper
@decorator
def test():
    print ("I am learning Decorators.")
test()
```

输出结果：

```
I learned Python Data Structures.
I am learning Decorators.
```

3.2.3 生成器

有时候集合的元素数量非常庞大，如果全部读取放入内存对硬件要求过大，此时我们可以使用生成器通过循环逐步获取集合后续的元素。

在 Python 中，使用了 yield 的函数被称为生成器（ generator ）。其基本用法如下：

1. 以列表推导式的形式创建生成器

使用圆括号编写生成器。

```
G = ( x*2 for x in range(5))                # 这里表示从 range(5)中每个 x*2
print(type(G))
for i in G:
    print (i)
```

输出结果：

```
<class 'generator'>
0
2
4
6
8
```

2. 使用 yield 创建生成器

使用 yield 关键字创建一个生成器，用以生成斐波那契数列。

```
def fib(n):
    current, num1, num2= 0, 0, 1                # 初始化数值从 0,1 开始，current 为循环
    while True:
```

```
        if current > n:                                  # 大于参数结束循环
            return
        yield num1                                       # yield 将函数变为一个生成器，函数不断循环
        num1, num2 = num2, num1+num2
        current += 1
fib(5)
for i in fib(5):
    print (i)
```

输出结果：

```
0
1
1
2
3
5
```

3. 唤醒生成器

可以使用 next 和 send 方法唤醒生成器，相比于 next 方法，send 在唤醒生成器时还可以向断点处传入一个数据。

```
def gen():
    i = 0
    while i<5:
        temp = yield i
        print(temp)
        i+=1
f = gen()
next(f)
>>>0
f.send('haha')
>>>haha
>>>1
next(f)
>>>None
>>>2
```

3.2.4 迭代器

迭代是 Python 非常强大的功能，它是访问集合的一种方式。迭代器是一个可以记住遍历位置的对象。迭代器从第一个元素开始访问，直到所有元素被访问完结束。

迭代器有两个基本的方法 iter()和 next()。迭代器示例如下：

1. 判断可迭代对象

迭代器只能向后。字符串、列表和元组对象都可以用于创建迭代器。数值不能迭代。

```
#使用 isinstance()方法判断对象是否是可迭代对象
from collections import Iterable                         # 可迭代对象
print(isinstance([], Iterable))
print(isinstance('abc', Iterable))
print(isinstance({'a':1,'b':2},Iterable))
print(isinstance(100, Iterable))
```

输出结果：

```
True
True
True
False
```

2. 迭代器基本方法

通过 iter()函数获取这些可迭代对象的迭代器。然后我们可以对获取到的迭代器不断使用 next()函数来获取下一条数据。

```
l = [1, 2, 3, 4, 5]
l_iter = iter(l)
next(l_iter)
>>>1
next(l_iter)
>>>2
```

3. 类实现迭代

类里实现__iter__()和__next__()方法，可以实现一个迭代器。

StopIteration 异常用于标识迭代的完成，防止出现无限循环的情况。

```
class Count:                          # 声明一个类实现迭代器
    def __iter__(self):               # 实现__iter__方法
        self.a = 1                    # 初始值为 1
        return self
    def __next__(self):               # 实现__next__方法，功能能为+1
        if self.a <= 5:
            x = self.a
            self.a += 1
            return x
        else:
            raise StopIteration       # 迭代完成
mycount = Count()
myiter = iter(mycount)
for i in myiter:                      # 使用 for 循环遍历迭代器
    print (i)
```

输出结果：

```
1
2
3
4
5
```

3.2.5 多任务

多任务并发编程是一个常见的功能，Python 的多任务分为多线程和多进程。

一个线程是完成特定任务的指令流，是处理器可以分配时间的最小执行单元。线程通常位于进程里面，由一个程序计数器、一个堆栈、一组寄存器和一个标识符组成。现在的 CPU 都能支持多线程处理任务。

一个进程里面包含一个主线程，还可以生成很多子线程。每个线程都包含自己的寄存器和堆栈等。我们可以有单线程的进程，也可以有多线程的进程。一般一个程序的执行实例就是一个进程。

3.2.5.1 多线程

Python 中多线程可以用 threading 模块实现。

1. 使用多线程执行任务

```
import threading
from time import sleep,ctime                    # sleep 可以指定程序等待时间，ctime 可以返回本地时间
def work1():                                    #  定义函数 work1，间隔 1 秒输出 work1 字符串
    for i in range(3):
        print("work1 正在执行...%d"%i)
        sleep(1)

def work2():                                    #  定义函数 work2，间隔 1 秒输出 work2 字符串
    for i in range(3):
        print("work2 正在执行...%d"%i)
        sleep(1)

if __name__ == '__main__':
    print('---开始---:%s'%ctime())

    t1 = threading.Thread(target=work1)  #  线程 1 调用 work1
    t2 = threading.Thread(target=work2)  #  线程 2 调用 work2
                                         #  启动线程
    t1.start()
    t2.start()

    sleep(5)
print('---结束---:%s'%ctime())
```

输出结果：

```
---开始---:Mon Apr 15 10:55:16 2019
work1 正在执行...0
work2 正在执行...0
work1 正在执行...1
work2 正在执行...1
work1 正在执行...2
work2 正在执行...2
---结束---:Mon Apr 15 10:55:21 2019
```

2. 线程同步

如果多个线程共同对某个数据修改，可能出现预料外的后果。为了保证数据的正确性，需要对多个线程进行同步。使用 Lock 可以实现简单的线程同步，这个对象有 acquire 方法和 release 方法。为了实现同步引入了锁的概念。锁有两种状态，锁定和未锁定。每当一个线程比如要访问共享数据时，必须先获得锁定。如果已经有别的线程获得锁定了，那么就此线程会暂停，也就是同步阻塞。等到其他线程访问完毕，释放锁以后，再让此线程继续。

示例：

```
import threading
import time
g_num = 0                          # 定义一个全局变量赋值为 0
def test1(num):                    # 定义函数 test1，对全局变量执行加操作
    global g_num                   # 使用全局变量
    for i in range(num):
        mutex.acquire()            # 上锁
        g_num += 1
        mutex.release()            # 解锁
    print("---test1--- g_num=%d" %g_num)

def test2(num):                    # 定义函数 test2，同样对全局变量加操作
    global g_num
    for i in range(num):
        mutex.acquire()            # 上锁
        g_num += 1                 # 执行加操作
        mutex.release()            # 解锁


    print("---test2--- g_num=%d" %g_num)
# 创建一个互斥锁
# 默认是未上锁的状态，可以删除锁后查看资源争夺的结果
mutex = threading.Lock()

# 创建 2 个线程，让他们各自对 g_num 加 1000000 次
p1 = threading.Thread(target=test1, args=(1000000,))
p1.start()

p2 = threading.Thread(target=test2, args=(1000000,))
p2.start()

# 等待计算完成
time.sleep(5)

print("2 个线程对同一个全局变量操作之后的最终结果是:%s" % g_num)
```

输出结果：

```
---test1--- g_num=1868536
---test2--- g_num=2000000
2 个线程对同一个全局变量操作之后的最终结果是:2000000
```

3.2.5.2 多进程

Python 要进行多进程操作，需要用到 muiltprocessing 模块。其中 Process 类和多线程很相似。

```
#coding=gbk                                    # 编码声明
from multiprocessing import Process
import os
import time

nums = [11, 22]                                # 设置全局的变量

def work1():
    """子进程 1 要执行的代码，对 nums 列表进行扩展"""
```

```
    print("in process1 pid=%d ,nums=%s" % (os.getpid(), nums))       # 获取进程号
    for i in range(3):
        nums.append(i)
        time.sleep(1)
        print("in process1 pid=%d ,nums=%s" % (os.getpid(), nums))

def work2():
    """子进程 2 要执行的代码，输出进程号和调用 nums 列表"""
    print("in process2 pid=%d ,nums=%s" % (os.getpid(), nums))

if __name__ == '__main__':
    p1 = Process(target=work1)
    p1.start()
    p1.join()                                          # 等此线程执行完后，再执行其他线程

    p2 = Process(target=work2)
    p2.start()
```

输出结果：

```
D:\Python Learning>python test.py
in process1 pid=23384 ,nums=[11, 22]
in process1 pid=23384 ,nums=[11, 22, 0]
in process1 pid=23384 ,nums=[11, 22, 0, 1]
in process1 pid=23384 ,nums=[11, 22, 0, 1, 2]
in process2 pid=20588 ,nums=[11, 22]
```

注：因为编辑器的缘故，Jupyter Notebook 不会输出本实验的结果。可以使用 Pycharm 执行或新建一个.py 文件执行。

华为认证系列教程

HCIP-Datacom-Network Automation Developer

Git操作

实验指导手册

版本:1.0

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： http://e.huawei.com

华为认证体系介绍

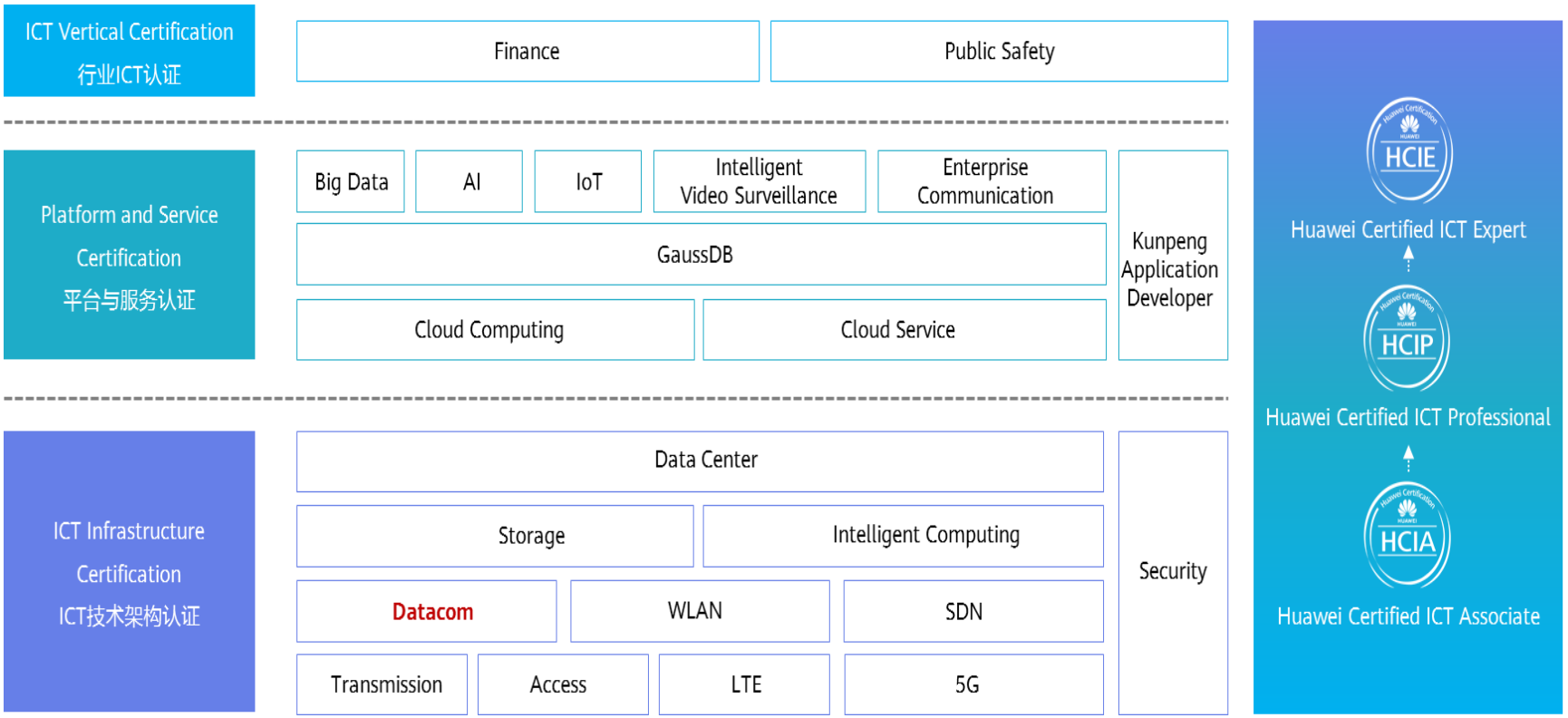
华为认证是华为公司基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，打造的ICT技术架构认证、平台与服务认证、行业ICT认证三类认证，是业界唯一覆盖ICT（Information and Communications Technology 信息通信技术）全技术领域的认证体系。

根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

HCIP-Datacom-Network Automation Developer定位于培养数通网络领域具备网络自动化开发专业知识和技能水平的高级工程师。通过HCIP-Datacom-Network Automation Developer认证将证明您能够胜任企业网络自动化开发工程师岗位，具备使用华为数通设备进行企业网络自动化部署、开发和运维的能力。

华为认证协助您打开行业之窗，开启改变之门，屹立在数通领域的潮头浪尖！

Huawei Certification



前 言

简介

本书为 HCIP-Datacom-Network Automation Developer 认证培训教程，适用于准备参加 HCIP-Datacom-Network Automation Developer 考试的学员，或者希望学习 Git 版本控制工具及华为云代码托管 CodeHub 基础知识和实践的读者。

读者知识背景

本文档主要适用于进阶学习的网络自动化工程师。读者需具备以下知识和技能：

- HCIA-Datacom

实验环境说明

环境介绍

本实验环境基于 Git 及其图形界面客户端 TortoiseGit，华为云代码托管服务 CodeHub。

本手册将介绍如何使用 Git 命令行方式和图形界面客户端 TortoiseGit 方式进行版本控制，并介绍华为云代码托管服务 CodeHub 的使用，最后对 Gitflow 工作流程进行实践。

使用说明

- 注册华为云帐号并实名认证：https://www.huaweicloud.com/。
- 访问华为云代码托管 CodeHub 成长地图：https://support.huaweicloud.com/codehub/index.html，获取更多信息。

目录

1 Git 基本操作实践

1.1 实验介绍

本实验使用命令行和图形界面客户端 TortoiseGit 两种方式介绍 Git 的基本操作——创建 Git 仓库、提交修改到本地 Git 仓库、查看历史提交记录、创建分支、合并分支和解决冲突。

通过本实验，您将学习：

- 创建本地 Git 仓库
- 实现提交代码修改到本地 Git 仓库
- 创建分支、合并分支和解决冲突
- 查找历史提交记录

1.1.1 实验步骤

本实验步骤如下：

1. 环境准备：本地安装 Git 和 TortoiseGit 图形界面客户端
2. 使用命令行方式进行 Git 基本操作
3. 使用 TortoiseGit 进行 Git 基本操作

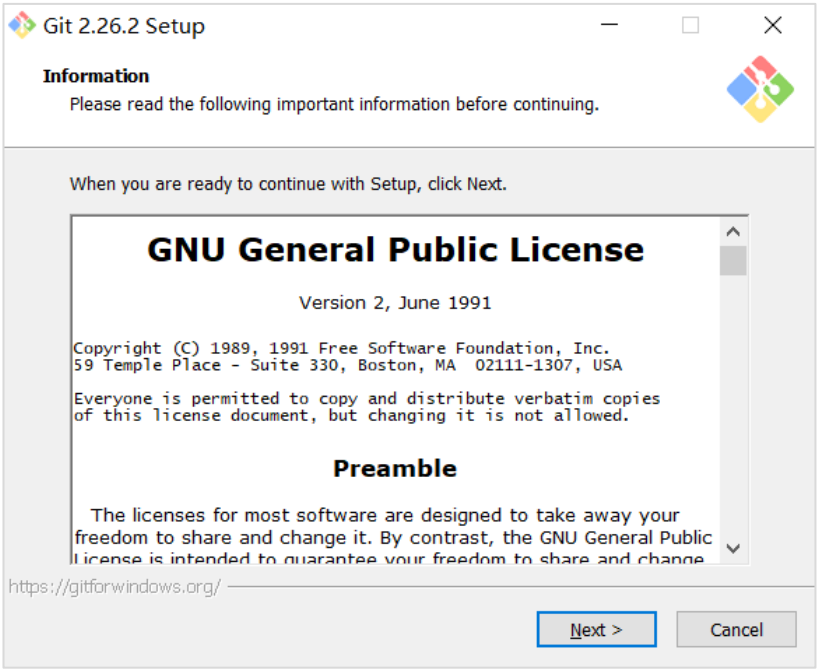
1.2 环境准备

1.2.1 Git 安装

1. 安装 Git

使用浏览器打开 Git 官网下载地址（https://git-scm.com/download/win）。根据您的操作系统位数下载 32 位/64 位的安装包。

双击运行安装包，在弹出的安装窗口中依次单击“下一步（Next）”，最后单击“安装（Install）”完成安装。



2. 配置 Git

单击 Windows“开始”图标，在“开始”搜索栏中输入“Git Bash”，单击回车即可打开 Git Bash 命令行终端。建议将其固定到 Windows 的任务栏中。

首先您需要配置用户名和邮箱，在 Git Bash 中输入以下命令行：

```
git config --global user.name "<您的用户名>"
git config --global user.email "<您的邮箱>"
```

说明：

- 用户名可以由字母、数字、常用符号组成；
- 邮箱请按照标准邮箱格式填写；
- git config 命令的–global 参数，表示你这台机器上所有的 Git 仓库都会使用这个配置，当然也可以对某个仓库指定不同的用户名和 Email 地址；

配置好之后可以使用以下命令行查看配置：

```
git config -l
```

生成 SSH 密钥，用来和 Git 远程仓库服务端进行鉴权认证，在 Git Bash 中输入以下命令行：

```
ssh-keygen -t rsa -C "<您的邮箱>"
```

然后输入 3 个回车（Enter 键）即可，生成的 SSH 秘钥对默认保存在“~/.ssh/id_rsa、~/.ssh/id_rsa.pub”。

```
$ ssh-keygen -t rsa -C "wjj1@huawei.com"                #生成 SSH 密钥
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/wjj1/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/wjj1/.ssh/id_rsa
Your public key has been saved in /c/Users/wjj1/.ssh/id_rsa.pub
The key fingerprint is:
```

```
SHA256:XsRZZsryXcV+9ruJo8QUsYtexviMF9rPFjgVdpI6iSM wjj1@huawei.com
The key's randomart image is:
+---[RSA 3072]----+
|                .+ ...|
|              o *o= o.|
|             . Bo+ =. |
|          E *+=oo   +|
|            So+B=   .o|
|           ...@o..   .|
|            .+ B. . .|
|               o oo...|
|                .o+.o.|
+----[SHA256]-----+
```

到这里您已经安装并配置了 Git。

1.2.2 TortoiseGit 安装

如果您不熟悉常用的 Git 命令，或者是从熟悉的 SVN 客户端（TortoiseSVN）迁移过来的，那么 TortoiseGit 客户端将是您更好的选择。TortoiseGit 是基于 Git 的图形界面客户端，需要先安装 Git 才能运行，请直接参照 1.2.1 章节进行安装。

1. 安装与第一次启动

1) 打开 TortoiseGit 官网 https://tortoisegit.org/download/下载链接，根据您的操作系统位数下载 32 位/64 位的安装包。

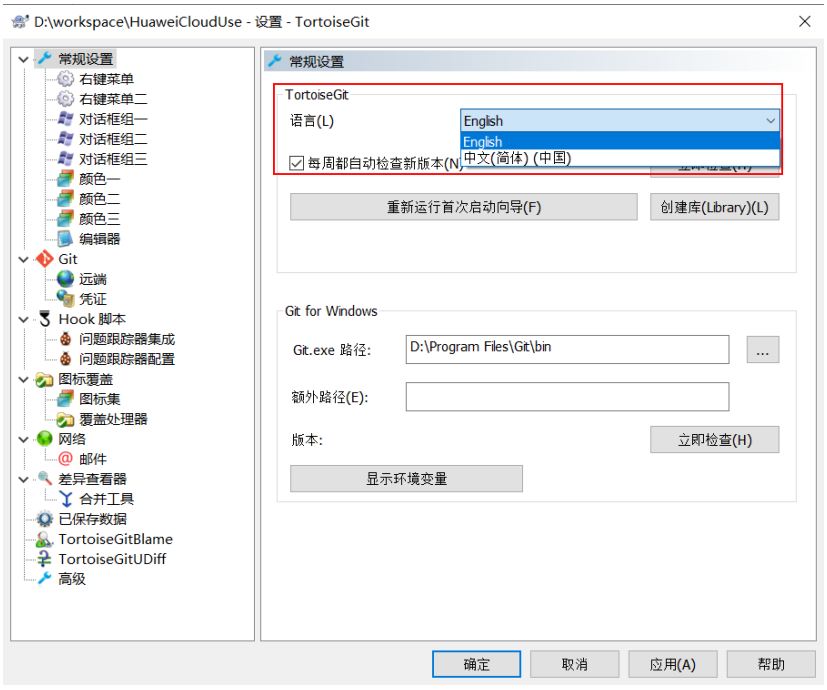
2) 双击运行安装包，在弹出的窗口中依次单击“Next”，然后单击“Install”即可完成安装，最后单击“Finish”即会运行第一次启动引导。

3) 在弹出的第一次启动引导中，会有 Language 语言选择、Git 可执行路径配置（自动填充可用的 Git 路径）、配置用户名和邮箱，保持默认依次单击 Next 完成即可。

2. 中文化（可选）

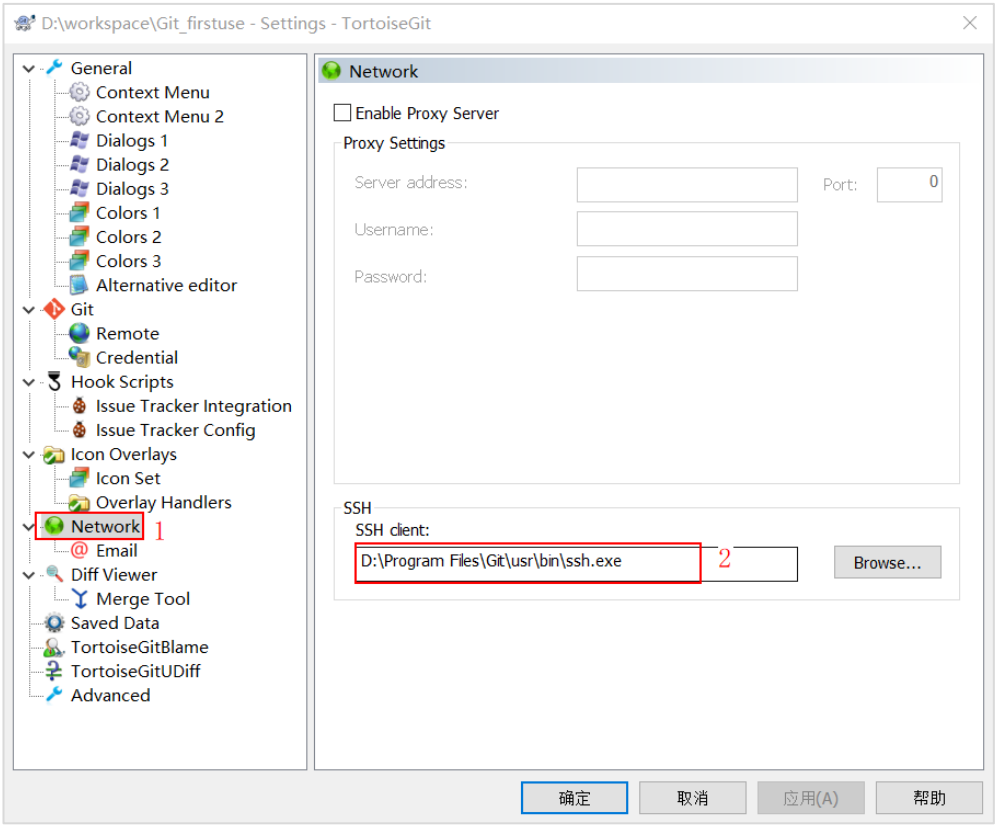
TortoiseGit 的安装包默认为英文，可以从 TortoiseGit 官网 https://tortoisegit.org/download/下载语言包（Language Packs），这里选择 Chinese simplified 语言包，下载对应的 32 位/64 位的语言包然后双击运行，在完成之前勾选“Configure TortoiseGit to use this language”即可完成汉化，如下图所示。

安装之后，在 TortoiseGit 的右键设置菜单的常规设置中，语言下拉框有 English 和中文（简体）两种选择。本实验为与命令行方式对应，TortoiseGit 语言选择 English。



3. 配置 TortoiseGit

TortoiseGit 同样需要一个密钥来和代码托管服务端进行鉴权认证，之前在 Git Bash 中已经生成了 SSH 密钥。将 TortoiseGit 设置的 Network 选项中的 SSH client 修改成 Git 安装目录下的 ssh.exe，即可直接使用 Git Bash 中生成的 SSH 密钥。



1.3 Git 命令行操作方式

本节介绍 Git 基本操作——创建本地 Git 仓库、提交修改到本地 Git 仓库、查看历史提交记录、创建分支、合并分支和解决冲突。与 Git 远程仓库交互在本实验手册第 2 节会结合华为云代码托管服务 CodeHub 介绍 Git 远程仓库交互相关操作。

1.3.1 创建本地 Git 仓库

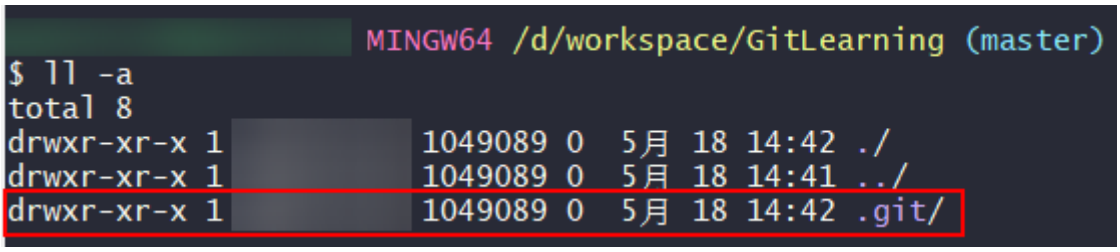
1. 使用命令行方式创建一个 Git 仓库非常简单，首先在本地 PC 选择一个合适的位置，创建一个空文件夹：

```
$ mkdir GitLearning                    #创建 GitLearning 文件夹
$ cd GitLearning/                      #进入 GitLearning 文件夹
```

2. 使用 git init 命令将该文件夹转换成 Git 可以管理的仓库：

```
$ git init                             #创建 Git 仓库
Initialized empty Git repository in D:/workspace/GitLearning/.git/
```

执行该命令后，Git 仓库就创建好了，并且是一个空的仓库（empty Git repository）。在原先的空文件夹下多了一个.git 文件夹，这个文件夹是 Git 用来跟踪管理版本库的。



1.3.2 提交修改到本地 Git 仓库

现在本地 Git 仓库已经创建完成，我们编写一个 readme.txt 文件，并将其添加到 Git 仓库中。使用如下命令：

```
$ vim readme.txt                    #创建 readme.txt 文件
```

输入以下内容：

```
This is a Git repository.
```

使用 git status 可以查看 Git 仓库状态。用来了解仓库中文件状态（可以查看到哪些文件被进行了修改，删除，增加等相关操作）。

```
$ git status                        #查看 Git 仓库状态
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        readme.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Untracked files 表示还没有被 Git 跟踪的文件。使用 git add 命令将文件添加到暂存区。

```
$ git add readme.txt                #将 readme.txt 文件添加到暂存区
warning: LF will be replaced by CRLF in readme.txt.
The file will have its original line endings in your working directory
```

继续使用 git status 查看 Git 仓库状态。

```
$ git status                        #查看 Git 仓库状态
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   readme.txt
```

Changes to be committed 表示待提交到 Git 仓库的修改。上面的输出表示新增一个 readme.txt 文件，但还没有提交到 Git 仓库。使用 git commit 提交修改。

```
$ git commit -m "add readme.txt file"                #将 readme.txt 文件提交到 Git 仓库
[master (root-commit) 28b687a] add readme.txt file
 1 file changed, 1 insertion(+)
 create mode 100644 readme.txt
```

-m 选项表示本次提交说明，这样就可以方便地从历史记录里找到这次提交记录。

1.3.3 查看历史提交记录

现在 Git 仓库里已经有了一次提交（readme.txt 文件），当提交次数达到一定数量，我们很难记得每次提交都修改了什么内容。使用 git log 命令可以查看历史提交记录，如果需要查看指定文件提交记录，可以在命令后接指定文件名称查看。

```
$ git log                                                    #显示历史提交记录
commit 28b687ac29cc4940710b3bb112b371c6ee46a260 (HEAD -> master)
Author: wjj1 <wjj1@huawei.com>
Date:    Mon May 18 16:35:43 2020 +0800

    add readme.txt file
```

git log 有许多附加选项，如--pretty==oneline 表示一行显示一条历史记录。

```
$ git log --pretty=oneline                                   #以 oneline 的格式显示历史提交记录
28b687ac29cc4940710b3bb112b371c6ee46a260 (HEAD -> master) add readme.txt file
```

更多 git log 信息可参考 https://git-scm.com/book/en/v2/Git-Basics-Viewing-the-Commit-History

1.3.4 创建分支

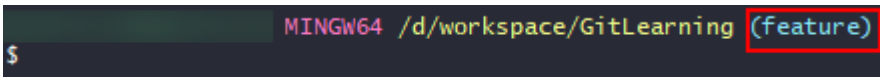
分支是 Git 的一个重要特性，使用分支意味着你可以把你的工作从主线上分离开来，转移到分支进行开发，以免影响主线开发。使用 git branch 命令创建分支，下面我们创建一个名为 feature 的分支，并在上面新增一些功能。

```
$ git branch feature                          #创建 feature 分支
```

将当前工作目录转移到 feature 分支。

```
$ git switch feature                          #切换到 feature 分支
Switched to branch 'feature'
```

命令行界面最右侧显示了当前工作分支，如下红框中的信息表示当前工作分支是 feature 分支。



在 feature 分支新增一个“helloworld.md”文件，代表新增的功能。

```
$ echo "hello world !" > helloworld.md                      #创建 helloworld.md 文件
$ git add helloworld.md                                     #将 helloworld.md 文件添加到暂存区
warning: LF will be replaced by CRLF in helloworld.md.
The file will have its original line endings in your working directory
$ git commit -m "add helloworld.md file"                    #将 helloworld.md 提交到 Git 仓库
[feature 9ce7bc3] add helloworld.md file
 1 file changed, 1 insertion(+)
 create mode 100644 helloworld.md
```

查看文件夹下内容，可以看到在 feature 分支下创建了一个新的 helloworld.md 文件。

```
$ ll                                          #显示文件夹内容
total 2
```

```
-rw-r--r-- 1 wWX935519 1049089 14   5 月  18 17:06 helloworld.md
-rw-r--r-- 1 wWX935519 1049089 26   5 月  18 15:02 readme.txt
```

1.3.5 合并分支

在 feature 分支完成功能开发后，将分支合并到主线（master）分支。

1. 将当前工作分支切换回 master 分支。

```
$ git switch master                          #切换到 master 分支
Switched to branch 'master'
$ ll                                         #查看 master 分支下内容，只有之前的 readme.txt 文件
total 1
-rw-r--r-- 1 wWX935519 1049089 26   5 月  18 15:02 readme.txt
```

2. 合并 feature 分支。

```
$ git merge feature
Updating 28b687a..9ce7bc3
Fast-forward
 helloworld.md | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 helloworld.md
```

git merge 命令的输出表示，新增了一个 helloworld.md 文件，这正是 feature 分支创建的文件。查看文件夹下内容，新增了 helloworld.md 文件。

```
$ ll                                         #查看文件夹内容
total 2
-rw-r--r-- 1 wWX935519 1049089 15   5 月  18 17:19 helloworld.md
-rw-r--r-- 1 wWX935519 1049089 26   5 月  18 15:02 readme.txt
```

1.3.6 解决冲突

冲突是合并分支时有可能遇到的情况。当合入分支和被合入分支同时对某一文件进行了修改，合并这两个分支时，就有可能会产生冲突而导致无法合并。这时就需要解决冲突。还是以 feature 分支和 master 分支为例。

1. 切换到 feature 分支，并对 helloworld.md 文件进行修改后提交 Git 仓库。

```
$ git switch feature
$ echo "This is feature branch." >> helloworld.md          #向 helloworld.md 文件写入内容
$ git add helloworld.md
$ git commit -m "Modify helloworld.md in feature branch."
```

2. 切换到 master 分支，并对 helloworld.md 文件进行修改后提交 Git 仓库。

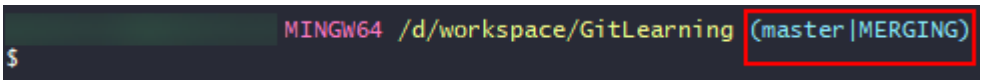
```
$ git switch master
$ echo "This is master branch." >> helloworld.md
$ git add helloworld.md
$ git commit -m "Modify helloworld.md in master branch."
```

3. 在 master 分支上合入 feature 分支。

```
$ git merge feature
Auto-merging helloworld.md
CONFLICT (content): Merge conflict in helloworld.md
```

```
Automatic merge failed; fix conflicts and then commit the result.
```

CONFLICT 字段表示合并分支时产生冲突，这时在命令行最右侧会显示有 MERGING 字段。



4. 解决冲突。

使用 git status 查看产生冲突的文件。

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   helloworld.md

no changes added to commit (use "git add" and/or "git commit -a")
```

上面表明 helloworld.md 文件产生了冲突，两个分支都对该文件进行了修改。需要手动进行修改以解决冲突。使用下面的命令打开冲突的文件。

```
$ vim helloworld.md                    #打开 helloworld.md 文件
hello world !
<<<<<<< HEAD
This is master branch.
=======
This is feature branch.
>>>>>>> feature
```

<<<<<<< HEAD 和=======之间的是 master 分支的修改，=======和>>>>>>> feature 之间是 feature 分支的修改。现在将文件内容手动修改成期望的内容，本实验中将其修改为如下内容：

```
hello world !
This is master branch.
```

输入:wq 保存并退出。

文件内容修改后，需要使用 git add 将该文件重新添加到暂存区，表示文件冲突已解决。

```
$ git add helloworld.md
```

然后输入 git commit，在弹出的窗口中输入:q，即可将因冲突而失败的合入重新操作完成。

```
$ git commit
[master e0cafd3] Merge branch 'feature'
```

使用 git log 带—graph 选项的命令，可以输出点线图清楚地看到分支合并过程。

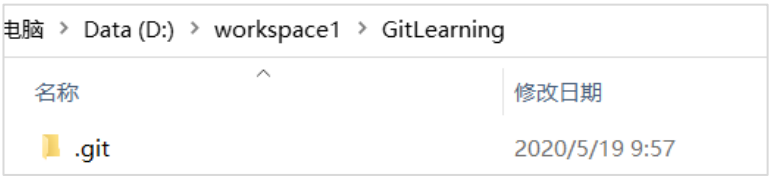
```
$ git log --pretty=oneline --graph
*   5719cfe6dcc9ea7588a980ab2bcbb95af45ef7e5 (HEAD -> master) Merge branch 'feature'
|\
| * c7687eea3e5730e47062e32abe17e3c84e928acc (feature) Modify helloworld.md in feature branch.
* | 85d92eb800f2d0f8ada4f279142ef0e5120f2148 Modify helloworld.md in master branch.
```

1.4 Git 图形界面操作方式

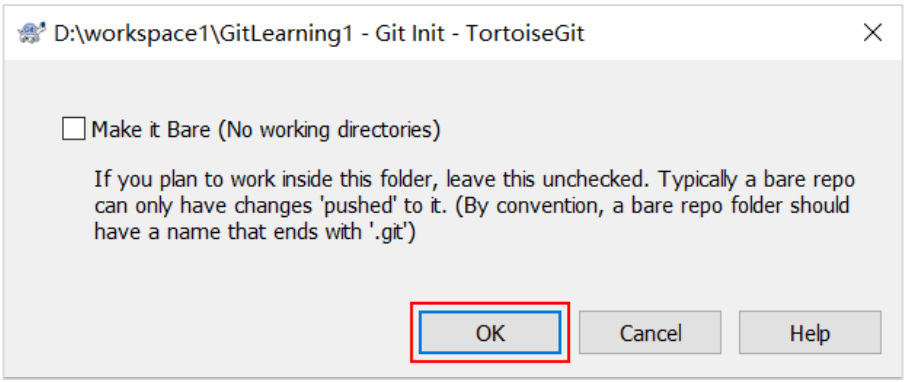
让我们用 TortoiseGit 将 Git 基本操作重新演练一次。

1.4.1 创建 Git 仓库

1. 使用 TortoiseGit 创建一个 Git 仓库，首先创建一个文件夹，并在该文件夹下面再创建一个名为.git 的文件夹，.git 文件夹是 TortoiseGit 创建的 Git 仓库所在的文件夹，外层文件夹是本地工作空间，Git 仓库会将文件检出到外层文件夹供开发人员进行编码工作。



2. 在上面的 GitLearning 文件夹空白处右键，单击“Git Create repository here...”，单击 OK 即可。



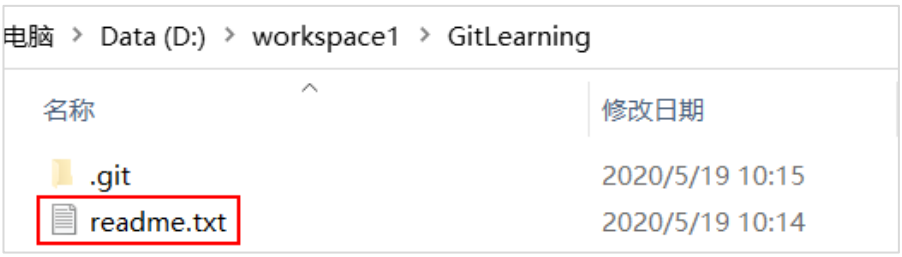
Git 仓库创建好之后，外层文件夹图标会有一个绿色圆形标志，这是 TortoiseGit 用来标识文件状态的标志，如已修改，冲突状态。这是相较 Git 命令行方式方便的一点，可以直观看到文件状态，而 Git 命令行方式需要输入 git status 才能查看文件状态。

之前创建的.git 文件夹，已经变成透明状态，并且 TortoiseGit 在.git 文件夹内生成了一些文件，将该文件夹初始化为 Git 仓库。

1.4.2 提交修改到本地 Git 仓库

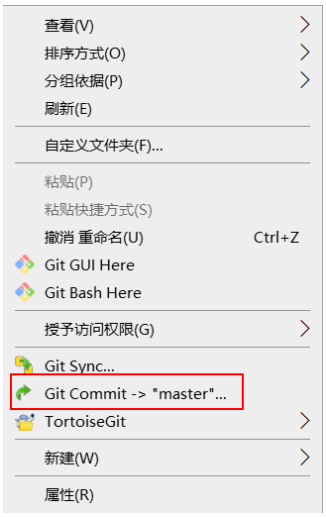
在 GitLeaning 文件夹下，用记事本创建一个文件，输入以下内容：

```
This is a Git repository.
```

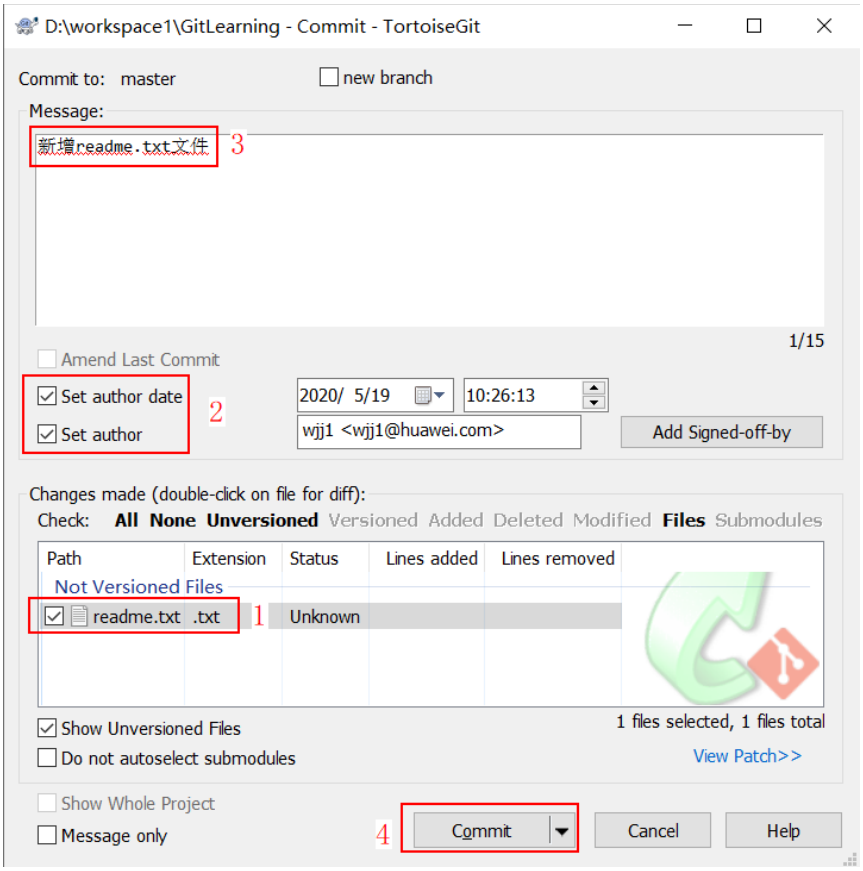


从上图可以直观地看出，readme.txt 没有被 Git 管理，因为该文件图标没有任何的状态标识。而 Git 命令行方式，需要使用 git status 查看文件状态，才能知道文件是否被 Git 管理。

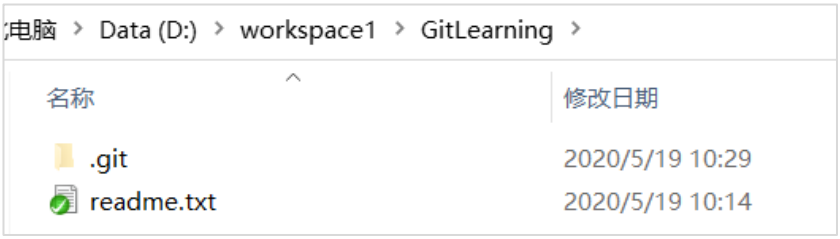
在文件夹空白处右键，选择“Git Commit”。



在弹出的界面中，按如下勾选，并在 Message 框中输入提交信息，再点击 Commit。

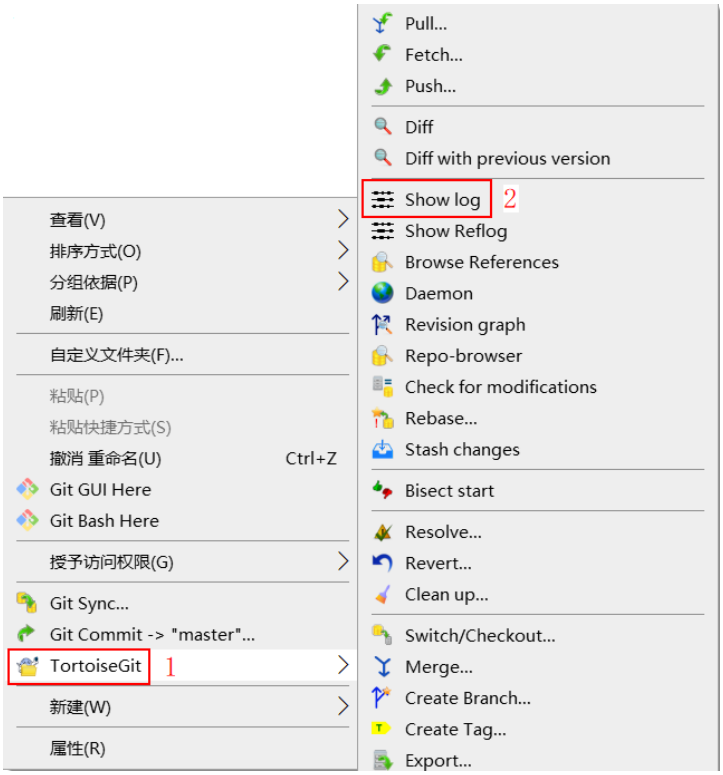


Commit 之后，readme.txt 已经提交到本地仓库，并且在图标上产生一个绿色标识，表明该文件已经是 Git 仓库中的最新版本。TortoiseGit 的 Commit 操作功能，其实是把命令行方式的 git add 和 git commit 命令合在了一起。



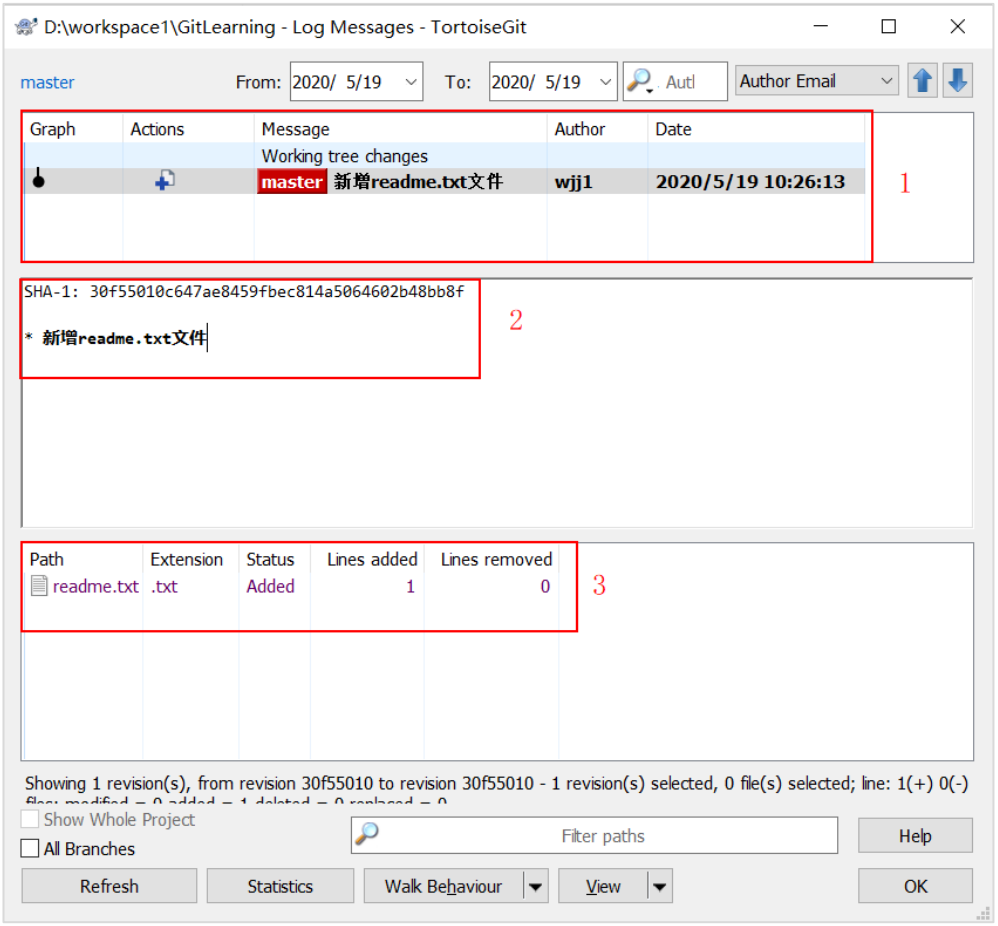
1.4.3 查看历史提交记录

现在仓库里已经有了一次提交（readme.txt 文件），当提交次数达到一定数量，我们很难记得每次提交都修改了什么内容。使用 TortoiseGit 可以方便地查看历史提交记录。



在弹出的界面中显示了历史提交记录，主要分为三个区域：

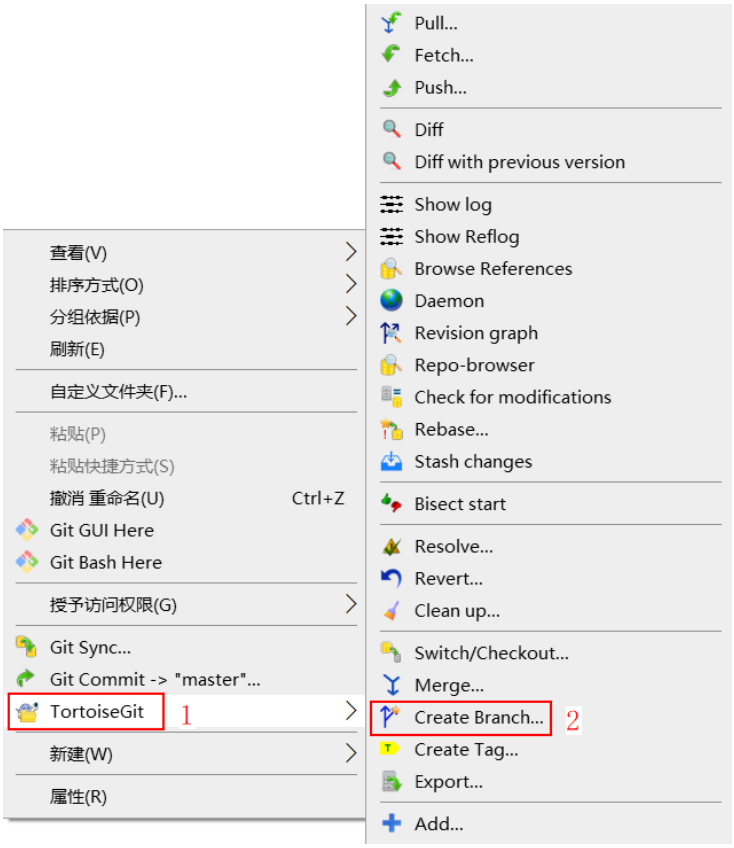
1. 历史提交记录的一个总结性的展示，包括提交时间、提交人和提交信息。
2. 对选中的某次提交记录的提交信息进行完整地显示。
3. 显示选中的提交记录所涉及修改的文件。



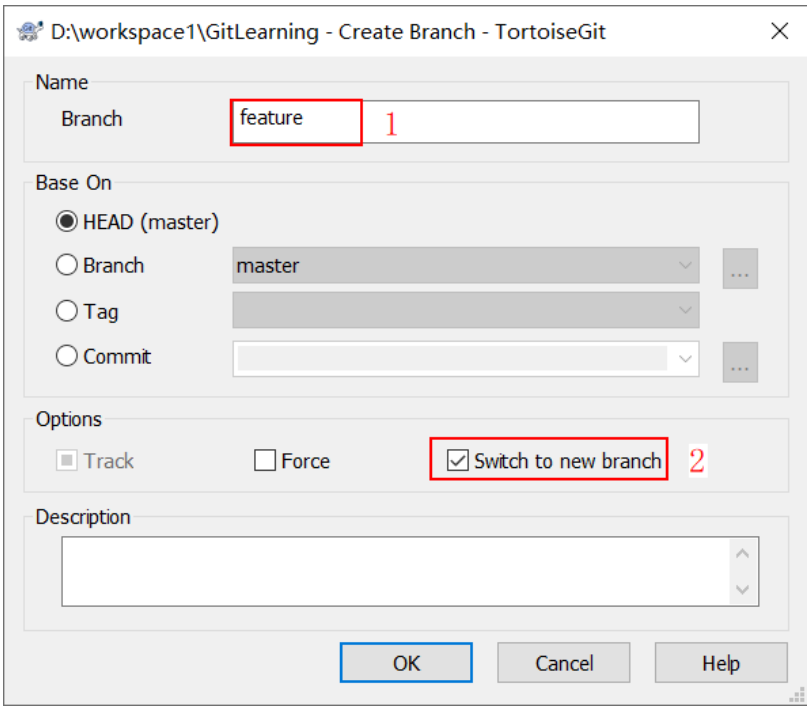
也可以在界面上方对提交记录根据提交人、提交信息、SHA1 等进行筛选。

1.4.4 创建分支

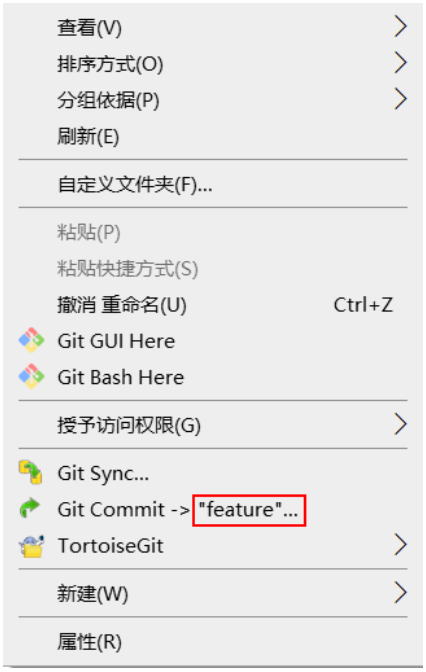
我们使用 TortoiseGit 创建一个名为 feature 的分支，并在上面新增一些功能。



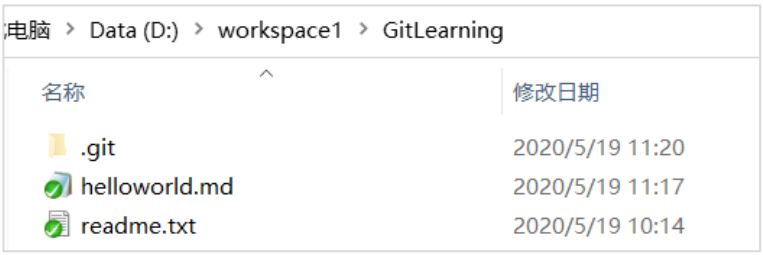
在弹出的界面中，输入分支名称 feature，并勾选“Switch to new branch”，这样创建了分支之后，就直接将工作分支切换为 feature 分支。



可以看到右键菜单中“Git Commit”后面跟的是“feature”，表示当前工作分支是 feature 分支。



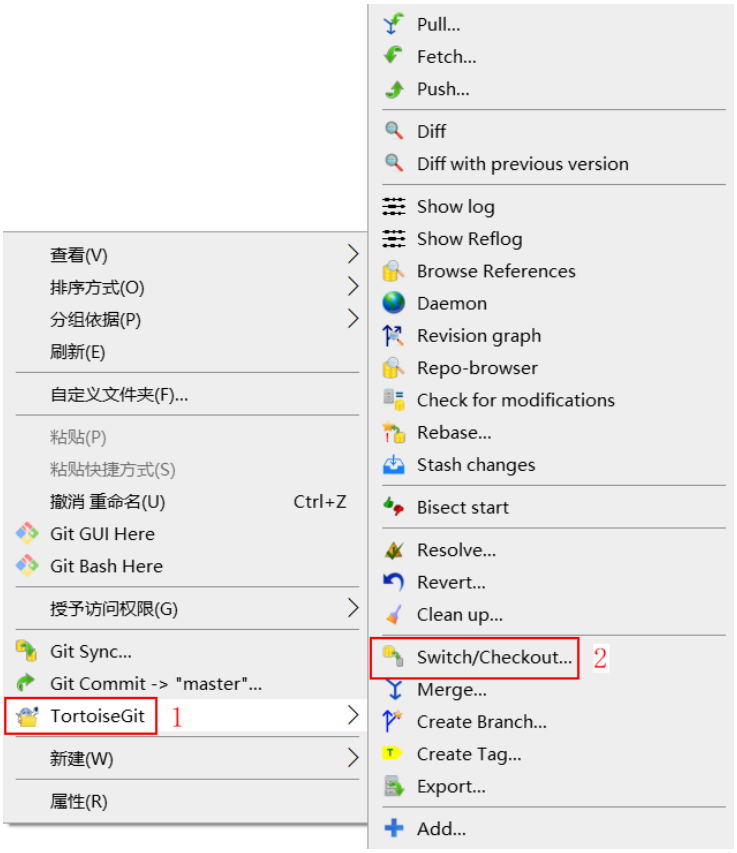
在 feature 分支工作空间下创建 helloworld.md 文件，代表新增的功能，并提交到 Git 仓库。因为涉及的操作都是之前小结介绍的操作，截图略。最后结果如下。

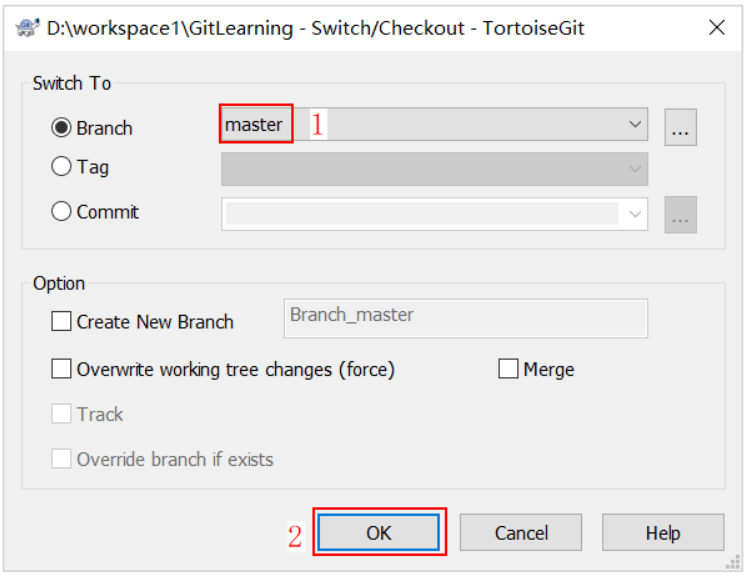


1.4.5 合并分支

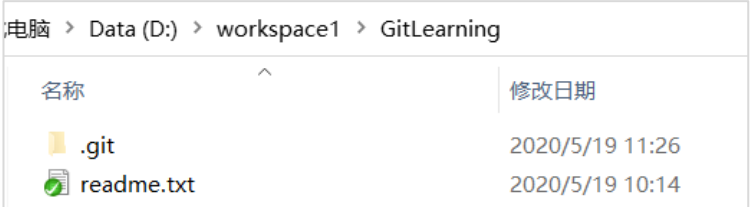
在 feature 分支完成功能开发后，将分支合并到主线（master）分支。

1. 使用 TortoiseGit 将分支切换回 master 分支。

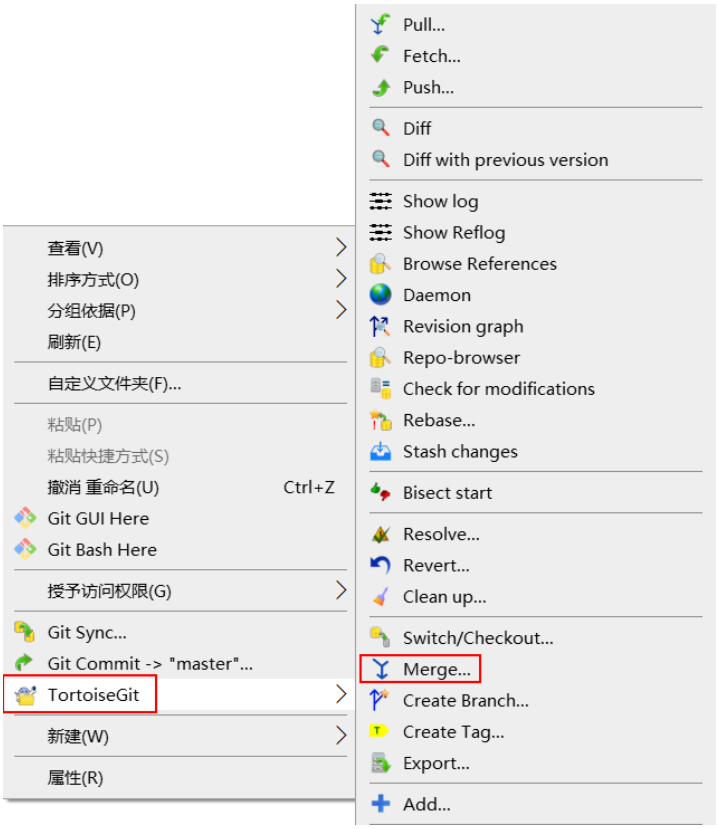




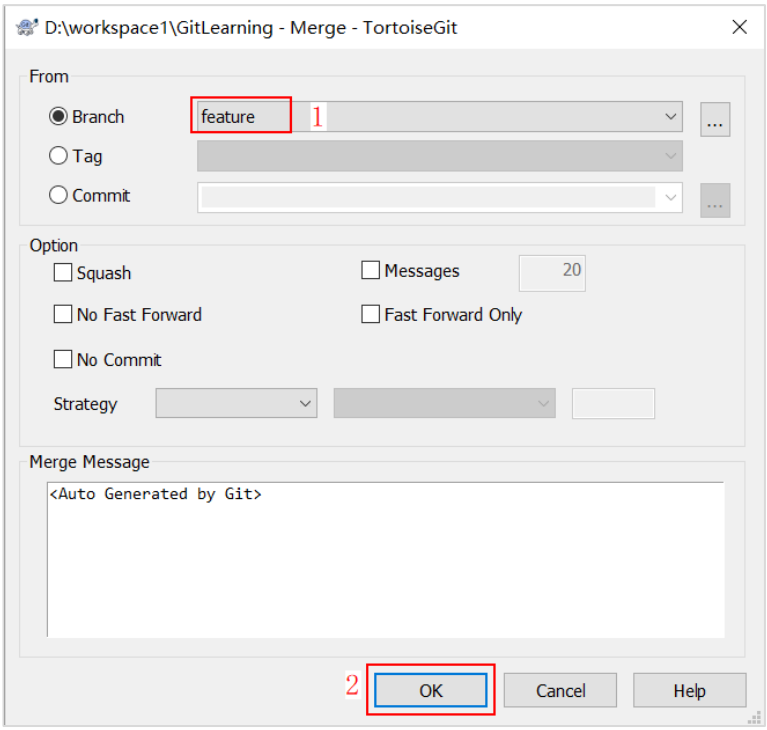
可以看到工作空间自动恢复到了之前 master 分支的状态，其中只有 readme.txt。



2. 合并 feature 分支。



在弹出的界面中选择要合入的分支 feature，然后点击 OK。

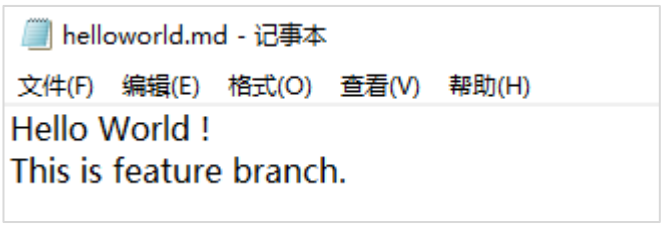


现在 master 分支的工作空间中已经有了 helloworld.md 文件，这正是 feature 分支创建的文件。

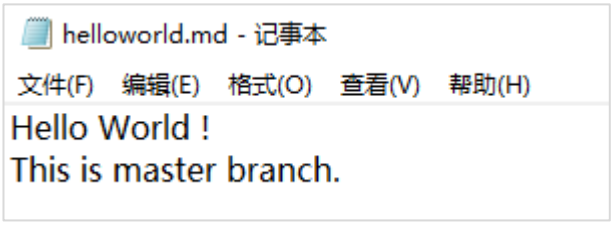
1.4.6 解决冲突

还是以 feature 分支和 master 分支为例，使用 TortoiseGit 进行冲突解决。

1. 切换到 feature 分支，并对 helloworld.md 文件进行修改后提交 Git 仓库。修改后内容如下。

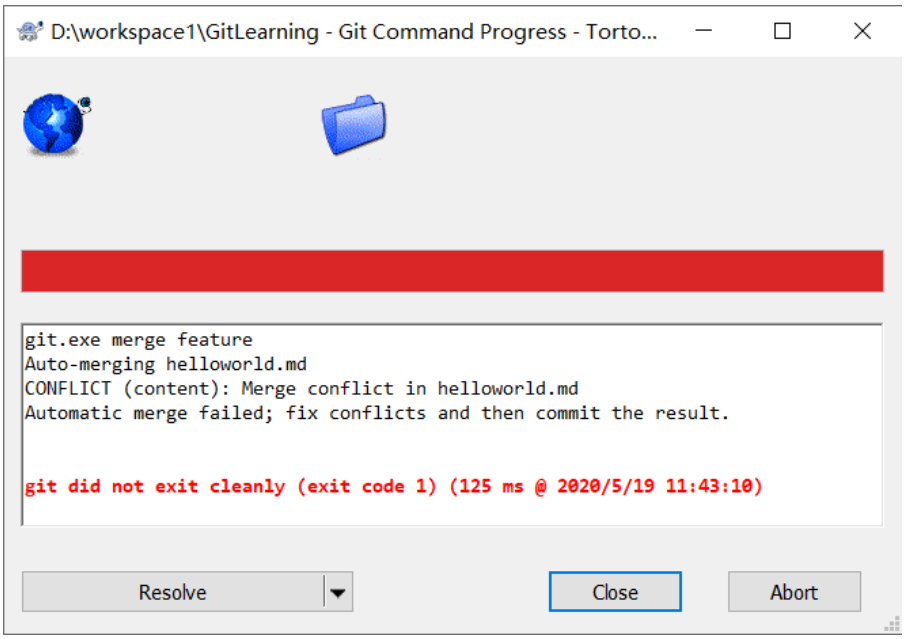


2. 切换到 master 分支，并对 helloworld.md 文件进行修改后提交 Git 仓库。修改后内容如下。



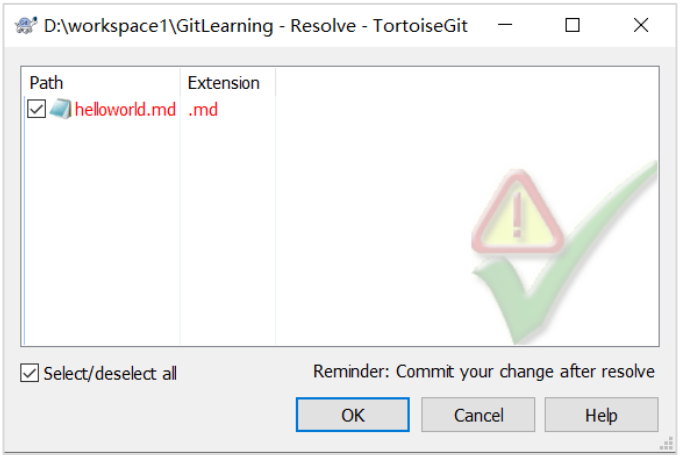
3. 在 master 分支上合入 feature 分支。

Merge 时会报产生冲突。

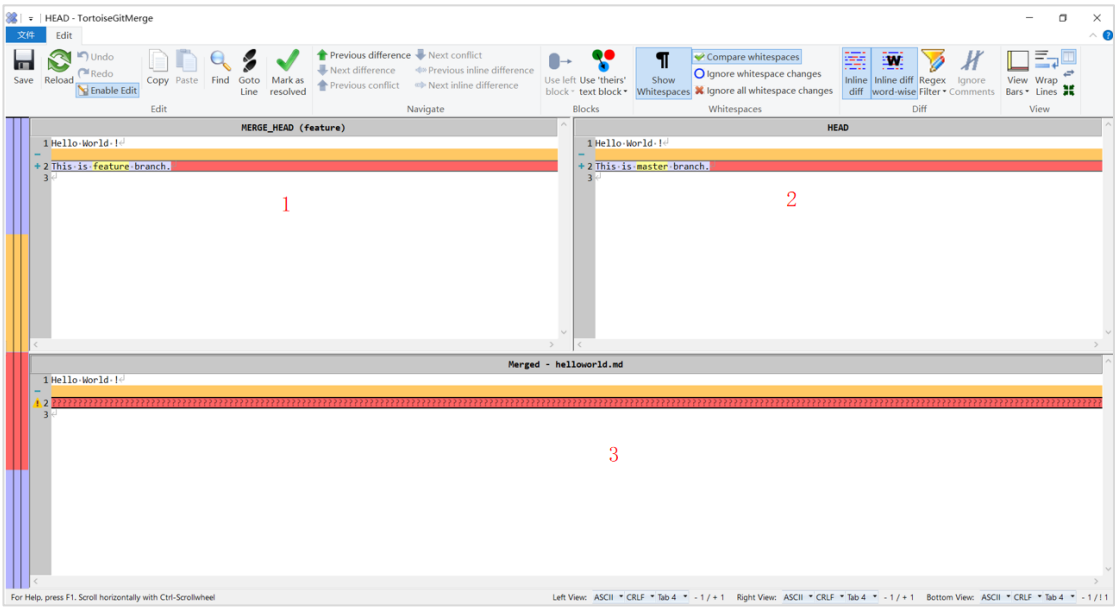


4. 解决冲突。

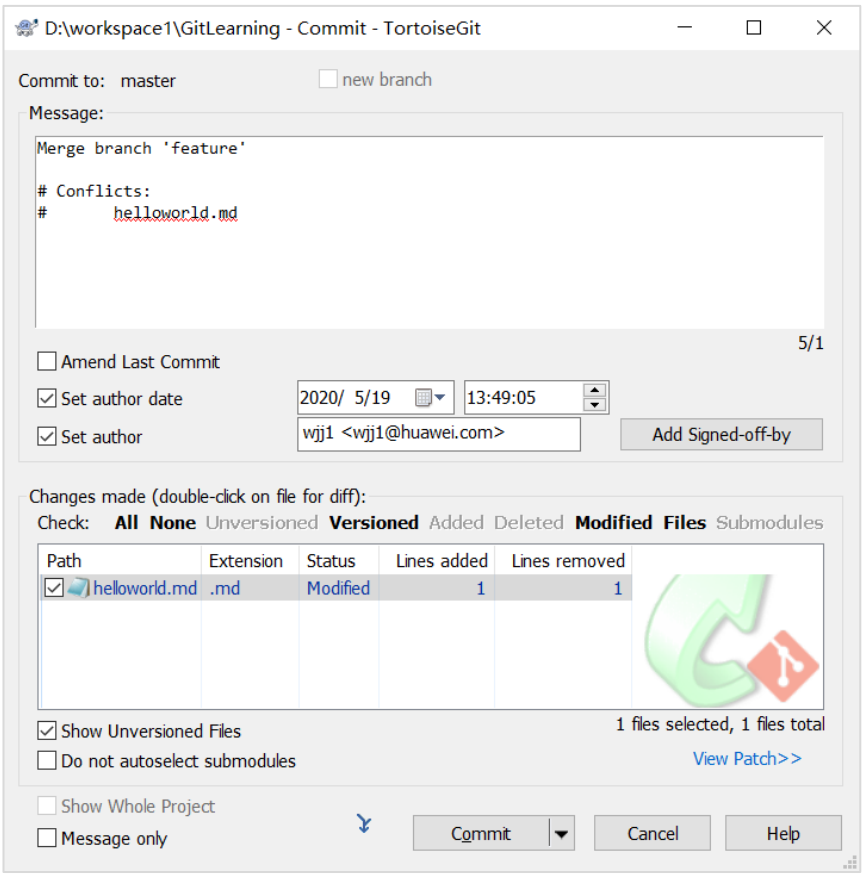
点击“Resolve”，弹出以下界面。里面列出了产生冲突的文件。



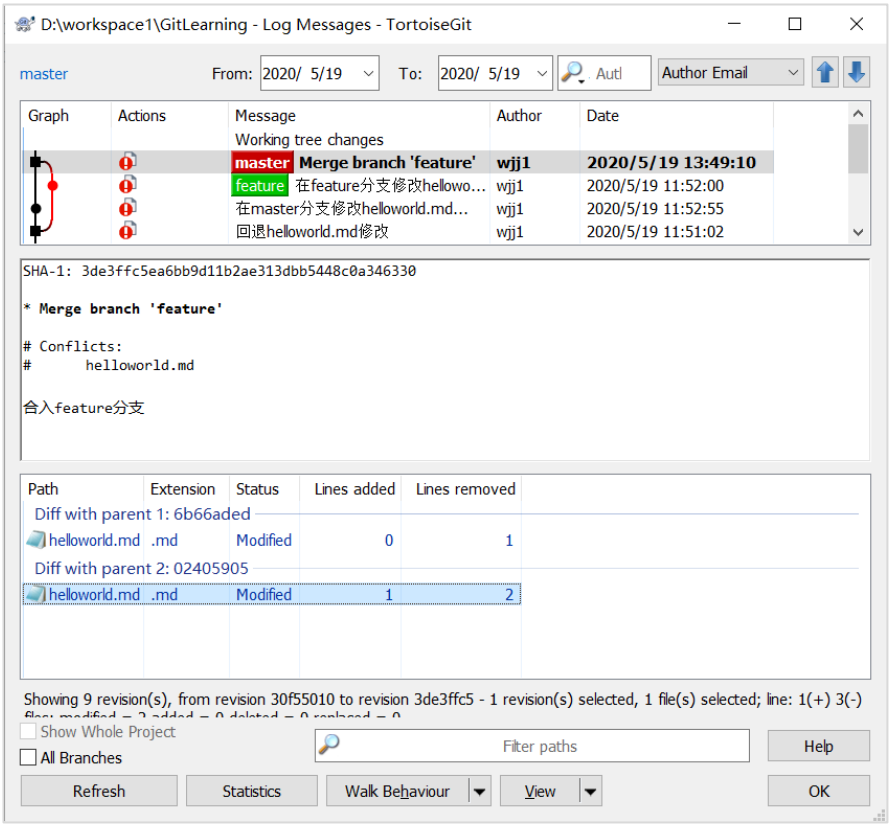
双击该文件，会以下界面。1 是 feature 分支的文件内容，2 是当前分支，也就是 master 分支的文件内容。3 是最后两个文件合并后的内容，因为产生冲突了，需要我们自己手动合并。在 3 中将合并内容修改完成后，点击上方工具栏的“Mark as resolved”，然后关闭窗口。



再在文件夹空白区域右键，点击“Git Commit”，会弹出下面界面，点击“Commit”提交即可，冲突就解决了，feature 分支成功合入 master 分支。



从提交历史记录也可以看出，feature 分支合入了 master 分支。



2 华为云代码托管实践

2.1 实验介绍

团队使用 Git 进行多人协作开发，需要一个 Git 远程仓库用于所有开发人员的代码提交和更新。华为云代码托管（CodeHub），是面向软件开发者的基于 Git 的在线代码托管服务，是具备安全管控、成员/权限管理、分支保护/合并、在线编辑、统计服务等功能的云端代码仓库，旨在解决软件开发者在跨地域协同、多分支并发、代码版本管理、安全性等方面的问题。

本实验将指导读者进行 CodeHub 常用操作，本课程您将会学习：

- 在 CodeHub 创建仓库
- 本地 Git 仓库推送代码到远程仓库
- 本地 Git 仓库从远程仓库更新代码

2.1.1 实验步骤

本实验步骤如下：

1. 环境准备：创建华为云账号并实名认证
2. 在 CodeHub 创建代码仓库
3. 使用 Git 命令行方式实现本地 Git 仓库与远程仓库的交互：克隆、推送、更新
4. 使用 TortoiseGit 实现本地 Git 仓库与远程仓库的交互：克隆、推送、更新

2.2 环境准备

1. 注册华为云帐号并实名认证

为保证帐户和资源安全，只有注册用户并且进行实名认证后才可以使用 DevCloud 服务。DevCloud 是集华为研发实践、前沿研发理念、先进研发工具为一体的软件开发平台，CodeHub 是其中的代码托管功能，为软件开发者提供基于 Git 的在线代码托管服务，包括代码克隆、下载、提交、推送、比较、合并分支、Code Review 等功能。

如果你没有注册，请按以下步骤进行：

1） 在浏览器上访问【华为云官网】 https://www.huaweicloud.com/

2） 填写个人信息， 并进行【实名认证】

3） 选择【个人账号】认证，用手机进行【扫码认证】

认证完成后，弹出绑定邮箱可忽略。

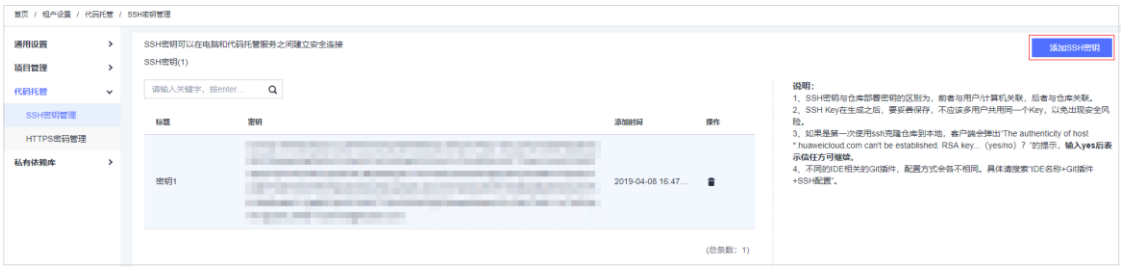
2. 添加 SSH 密钥到代码托管服务端

1) 打开 Git bash 客户端，将 SSH 密钥“~/.ssh/id_rsa.pub”的内容打印出来，如下图所示。

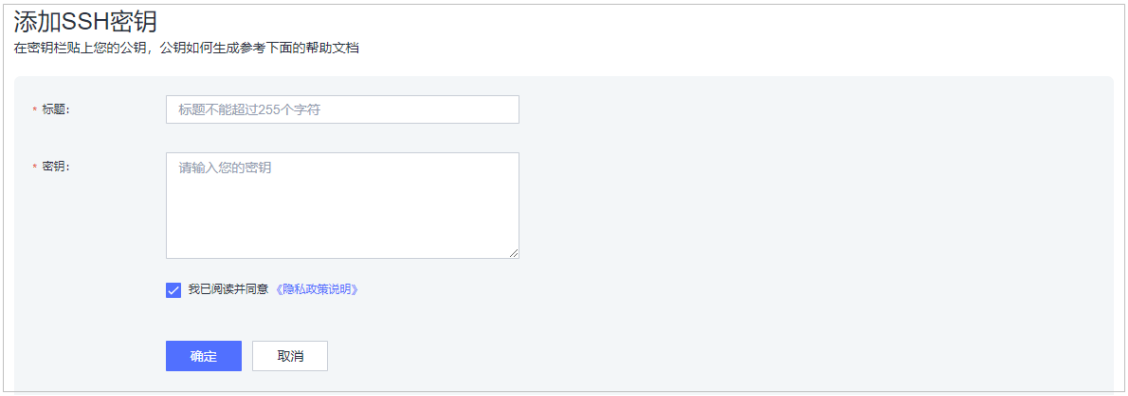
```
$ cat ~/.ssh/id_rsa.pub                    #输入 id_rsa.pub 内容
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQDKvEG8tcNEYW8hdTQrtw51ZF6Hpynj2qxUAGEBovdhHAoRsxfD
pumtKe88pRTqWk08gtoaVK54niNOBEE4jSP73nKdsajCoXlrnaVC/TBTxW+plRFoAk+DSYJEkWLK/aBk7sjDW
sLiZmU2z0M0ifx/Vn5tyyc5JC+IijZfuHX4r11P03DfrXXn21lfYfcXFXyvDrA9ojFlxgX0Xuc53vo1nUcsng+hqQuXf
RLe/mT4nQesS81BD3TAYOv5UUDVj60CdMW2ZqULCl6Yez/OvLWT2h9r4y2zG1DEOhhyDxXM++Z5HIE+q
A11BKUFGPFQClL2BiNFRuTPag9dBAfeMhAjOVHW30pPEWK+BCZgVyH/SQ4tu9xr3WGxJMmjaoQ+VOOl/
u8B7oKI4iw4rOAYekv0HvFdXlfl3a9qJjdx+v1vkL9lss7wsqVpf6tX5IAzglfZzsNWMOKx1KksbVGAXAcKONsHX
DxyogXxvQ1h/Vmh4pZoTAwfLhpBNcvejLvblic= wjj1@huawei.com
```

复制上述的 SSH 密钥内容，进入代码托管首页 https://devcloud.huaweicloud.com/codehub/home，单击“设置我的 SSH 密钥”按钮进入 SSH 密钥管理页面，进行添加。

进入 SSH 密钥管理页面，如下图所示。



2) 单击“添加 SSH 密钥”按钮进行添加，粘贴上述复制的 SSH 密钥内容、填写标题，单击“确定”即可，如下图所示。



3) 至此，您已经设置好了 SSH 密钥，您可以继续设置 HTTPS 密码。

3. 设置 HTTPS 密码

HTTPS 密码是使用 HTTPS 协议和代码托管服务端交互的凭证，默认与华为云登陆密码一致。如需修改，修改步骤如下：

1) 进入代码托管首页，单击“设置我的 HTTPS 密钥”，显示“HTTPS 密钥管理”页面。单击自行设置密码。

2) 在弹出的界面中，单击“修改”，修改密码需要绑定邮箱。绑定邮箱后，重新设置密码后点击“保存”即可。

3) 到这里您已经完成了 HTTPS 密码设置。可以在 CodeHub 创建一个仓库了。

2.3 CodeHub 代码仓库创建

2.3.1 创建项目

代码仓库的创建依托于项目的创建，因此需要先创建一个项目。

1. 选择资源存储区域。

1) 访问 https://devcloud.huaweicloud.com/home，进入 DevCloud 主页面。选择右上角的区域，在下拉菜单中选择“华北-北京四”。

2. 新建 Scrum 项目

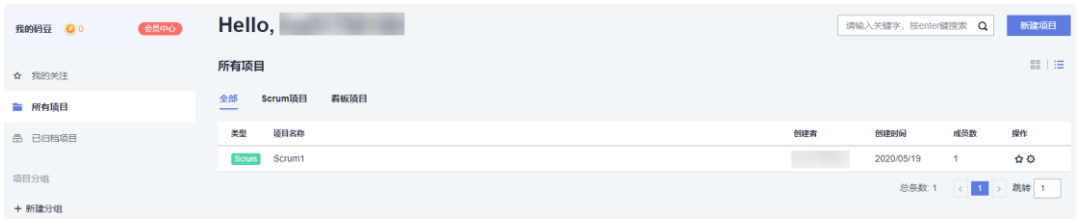
1) 单击首页右上角区域的“新建项目”按钮。

2) 单击 Scrum 卡片，选择 Scrum 流程。

3) 填写项目名称和项目描述。

4) 单击“确定”后，进入当前工作空间中。

至此，您已经拥有一个网络开发环境。完成项目创建后，您可以在首页所有项目列表中查看新建项目，并进入该项目工作区。



2.3.2 创建代码仓库

项目创建完成后，即可在该项目下进行代码仓库的创建。

1. 切换到代码托管页面

若已经进入项目工作空间，单击菜单栏“代码 > 代码托管”，切换至“代码托管”页面。

或者在首页所有项目列表中选择一个项目，鼠标移至该项目，单击“代码”，进入“代码托管”页面。

2. 使用普通新建模式创建代码仓库

1） 点击右上角“普通新建”按钮，选择“普通新建”

2) 输入代码仓库名称、描述等信息后点击“确定”

至此，您已经创建好了一个代码仓库。

新建的代码仓库只有一个自带的 README.md 文件。

现在，可以从本地克隆远程仓库，进行多人协同开发了。

2.4 本地 Git 仓库与远程仓库交互

本节使用 Git 命令行方式和 TortoiseGit 图形界面方式实现本地 Git 仓库与远程仓库的基本交互操作：克隆、推送、更新。

2.4.1 Git 命令行操作方式

2.4.1.1 克隆

1. 进入代码仓库，点击“克隆/下载”按钮，复制 SSH 克隆地址（也可以使用 HTTPS 克隆地址，本手册使用 SSH）。

2. 在选定的文件夹打开 git bash，输入 git clone 命令进行克隆。

```
$ git clone git@codehub.devcloud.cn-north-4.huaweicloud.com:Scrum100005/ScrumDev.git
Cloning into 'ScrumDev'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (3/3), done.
```

git clone 命令可以在仓库地址后加文件夹名称参数，表示本地文件夹的名称。如下面的命令表示克隆远程仓库到本地 PC 的 MyDir 文件夹。不带参数的情况下，本地文件夹名称默认与远程仓库名称相同。

```
$ git clone git@codehub.devcloud.cn-north-4.huaweicloud.com:Scrum100005/ScrumDev.git MyDir
```

查看文件夹内容，可以看到远程仓库已经克隆到了本地 PC。

```
$ ll                                    #查看文件夹内容
total 1
drwxr-xr-x 1 wWX111111 1049089 0   5 月  19 16:51 ScrumDev/
```

2.4.1.2 推送

我们在本地新增一些功能，并推送到远程仓库。增加一个 helloworld.py 文件。

```
$ vim helloworld.py
```

输入如下内容（按字母键 i 后可以开始输入），并输入:wq 保存并退出。

```
print("Hello World !")
```

先将 helloworld.py 提交到本地仓库。

```
$ git add helloworld.py
$ git commit -m "add helloworld.py"
```

再使用 git push 推送到远程仓库。

```
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 301 bytes | 301.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To codehub.devcloud.cn-north-4.huaweicloud.com:Scrum100005/ScrumDev.git
   e8ca831..44391be  master -> master
```

打开远程代码仓库，可以看到 helloworld.py 文件已经被推送上来了。

2.4.1.3 更新

假设现在有另一名开发人员 B，修改了 helloworld.py 文件，并推送到了远程仓库。我们需要将 B 的修改更新到本地。另一名开发人员我们可以通过在本地创建另一个仓库来模拟。

```
$ git clone git@codehub.devcloud.cn-north-4.huaweicloud.com:Scrum100005/ScrumDev.git ScrumDev_B
$ cd ScrumDev_B/
$ vim helloworld.py
```

修改成以下内容，保存后退出。

```
print ("Hello World !\nI'm xxx")
```

先将修改后的 helloworld.py 提交到本地仓库，并推送到远程仓库。

```
$ git add helloworld.py
$ git commit -m "Modify helloworld.py by developer B"
$ git push
```

现在我们需要使用 git pull 命令将开发人员 B 的修改更新到本地仓库。

```
$ cd ../ScrumDev                                    #进入 ScrumDve 本地仓库所在文件夹
$ git pull                                          #更新本地主线分支
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), 306 bytes | 12.00 KiB/s, done.
From codehub.devcloud.cn-north-4.huaweicloud.com:Scrum100005/ScrumDev
   44391be..c618432  master     -> origin/master
Updating 44391be..c618432
Fast-forward
 helloworld.py | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

打开 helloworld.py 文件，可以看到内容已经更新成开发人员 B 修改后的状态。

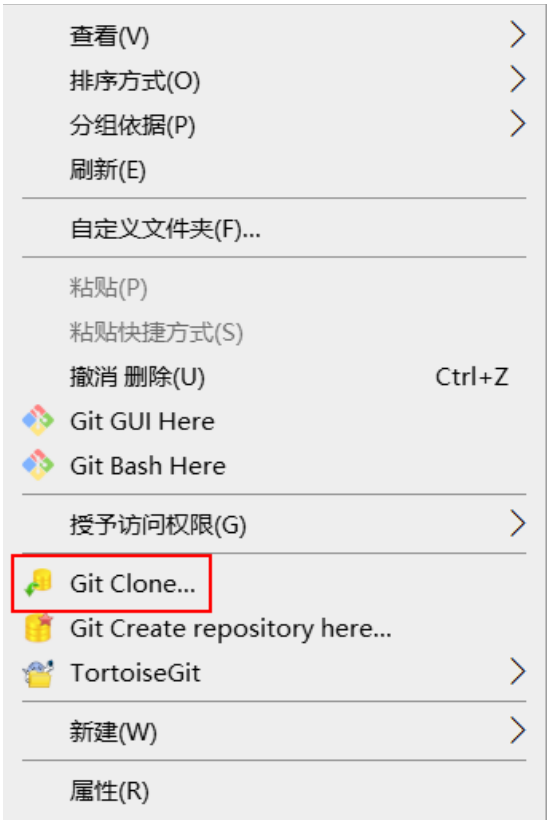
```
$ vim helloworld.py
print("Hello World !\nI'm xxx")                    #打开后，helloworld.py 的文件内容
```

2.4.2 Git 图形界面操作方式

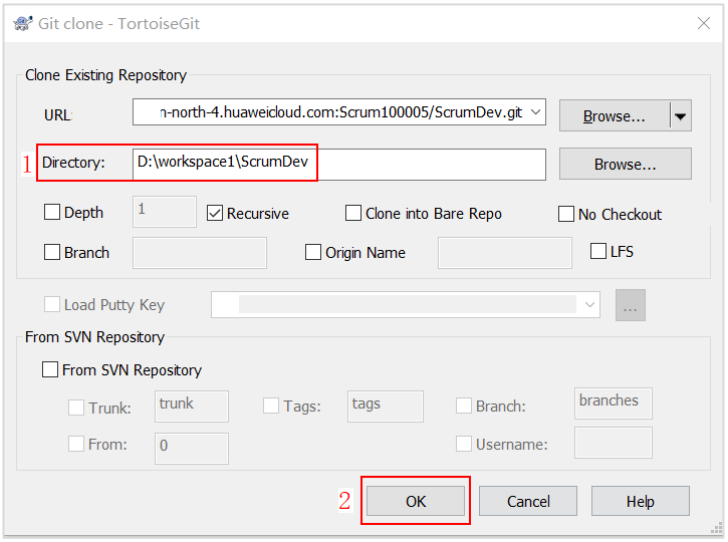
我们将克隆、推送和更新的操作，用 TortoiseGit 重新操作一遍。

2.4.2.1 克隆

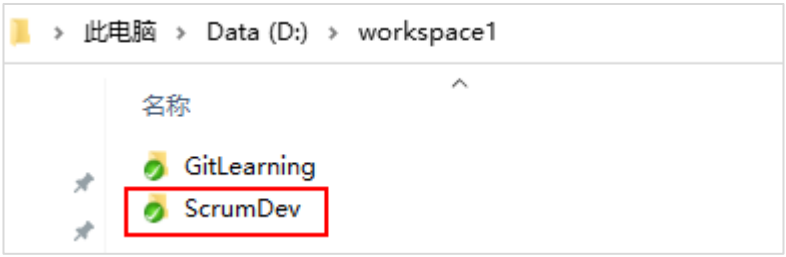
1. 进入代码仓库，点击“克隆/下载”按钮，复制 SSH 克隆地址。
2. 在目标文件夹空白处右键，单击“Git Clone”。

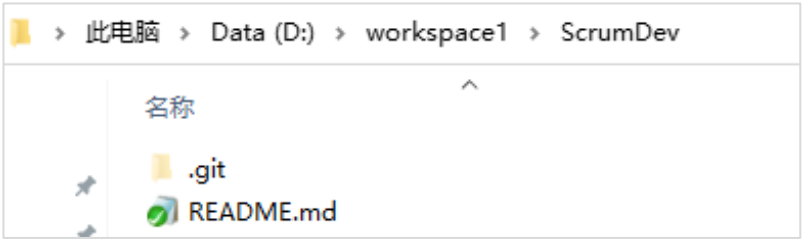


在弹出的窗口中，输入本地仓库所在文件夹地址，默认本地文件夹名称和远程仓库名称相同。然后点击 OK。



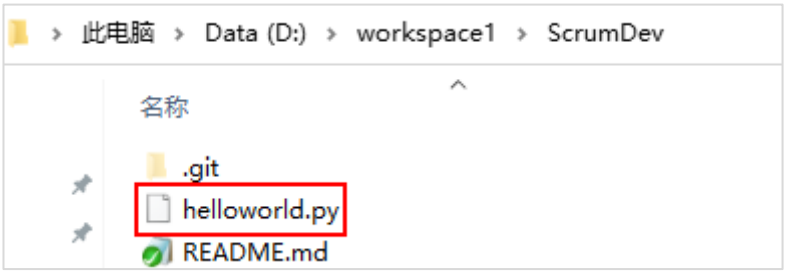
可以看到远程仓库已经被克隆到本地。





2.4.2.2 推送

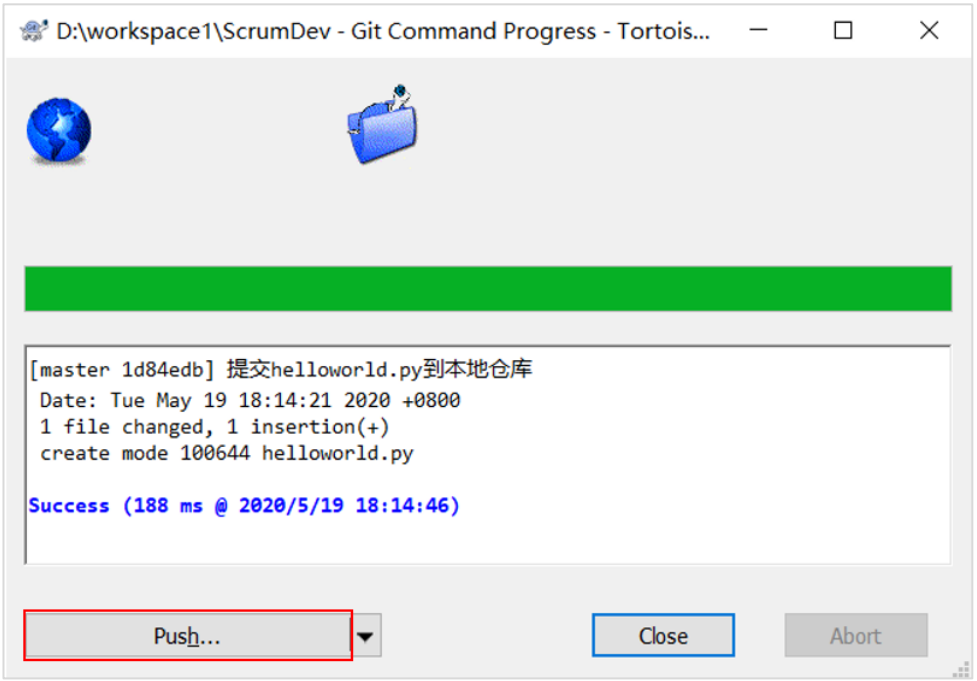
我们在本地新增一些功能，并推送到远程仓库。增加一个 helloworld.py 文件代表新增的功能。



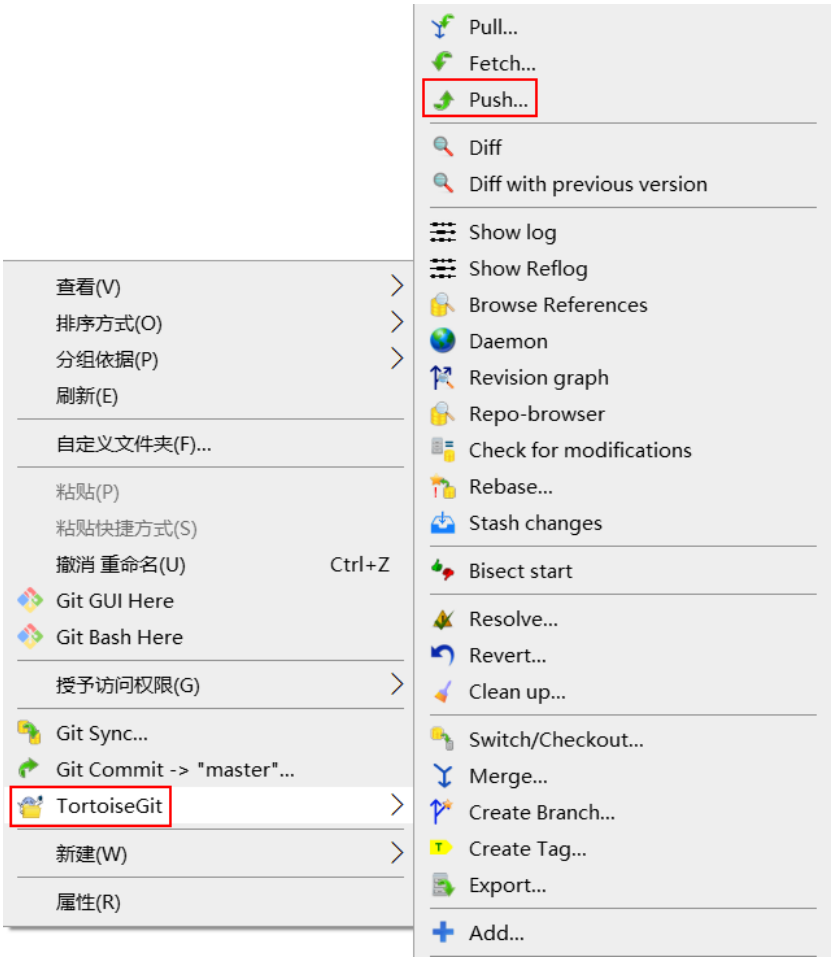
文件内容如下。

```
print("Hello World !")
```

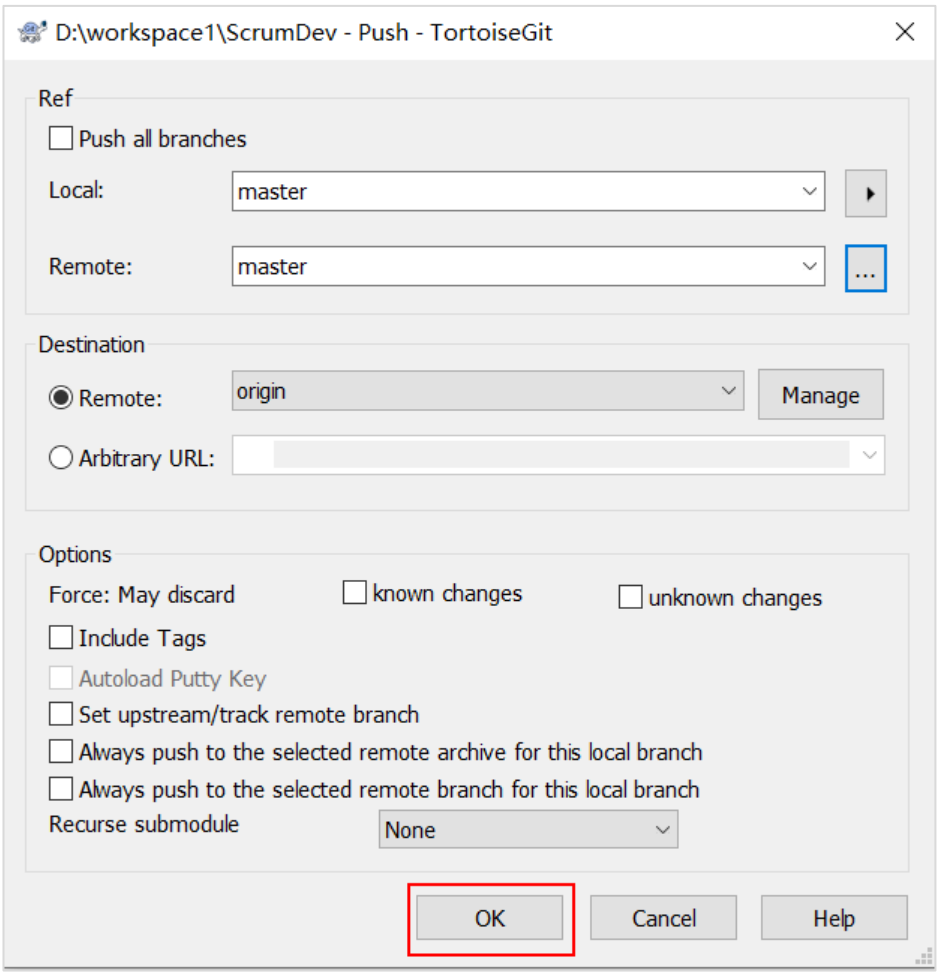
将 helloworld.py 提交到本地仓库（提交的操作在 1.3.2.2 已经介绍，此处不再赘述）。点击“Commit”后，会弹出如下界面。直接点击“Push”，即可将修改推送到远程仓库。



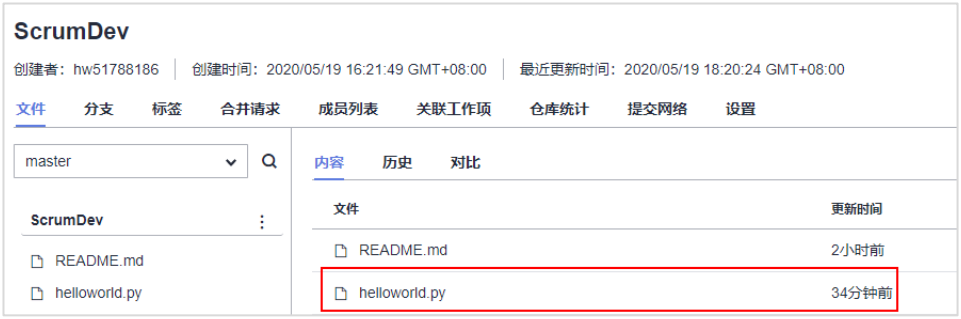
如果不想立即推送到远程仓库，可以点击“Close”关闭。之后在文件夹右键选择“Push”，



在弹出的界面选择 OK 即可。

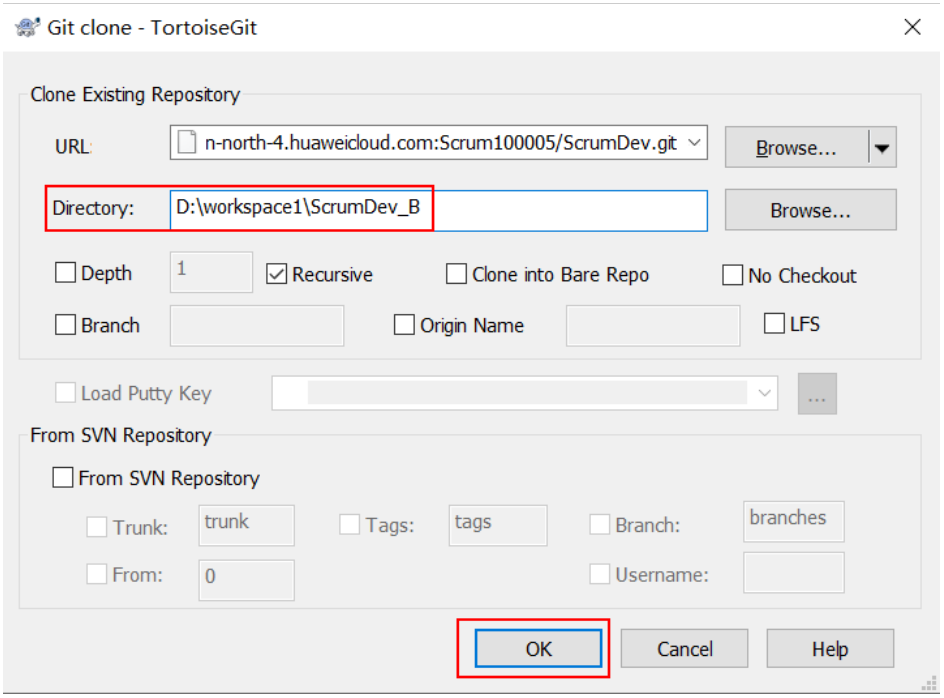


打开远程代码仓库，可以看到 helloworld.py 文件已经被推送上来了。



2.4.2.3 更新

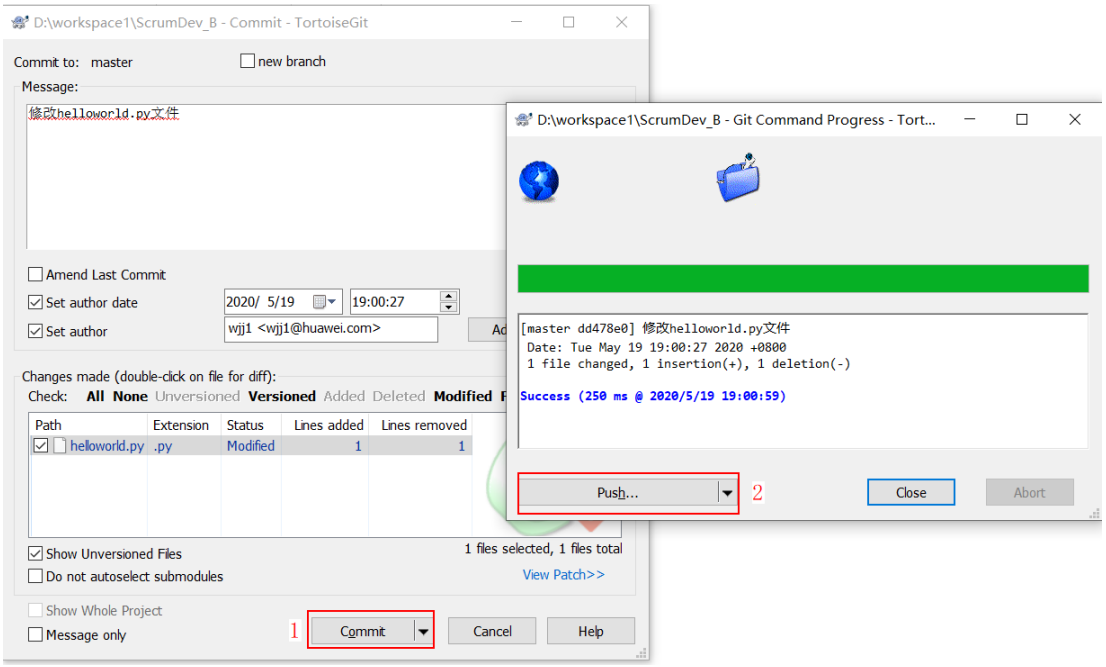
假设现在有另一名开发人员 B，修改了 helloworld.py 文件，并推送到了远程仓库。我们需要将 B 的修改更新到本地。另一名开发人员我们可以通过在本地创建另一个仓库来模拟。在本地 PC 使用 TortoiseGit 克隆远程仓库到 ScrumDev_B 文件夹。



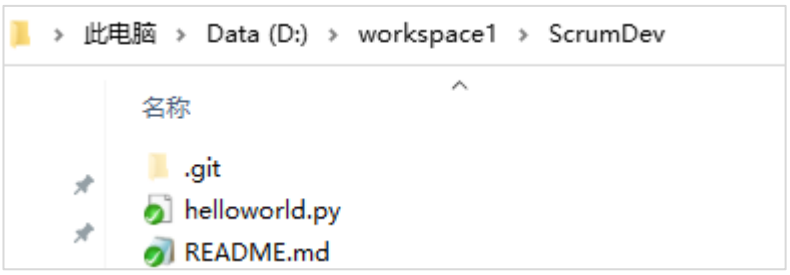
将 helloworld.py 文件修改成以下内容。

print("Hello World !\nI'm xxx")

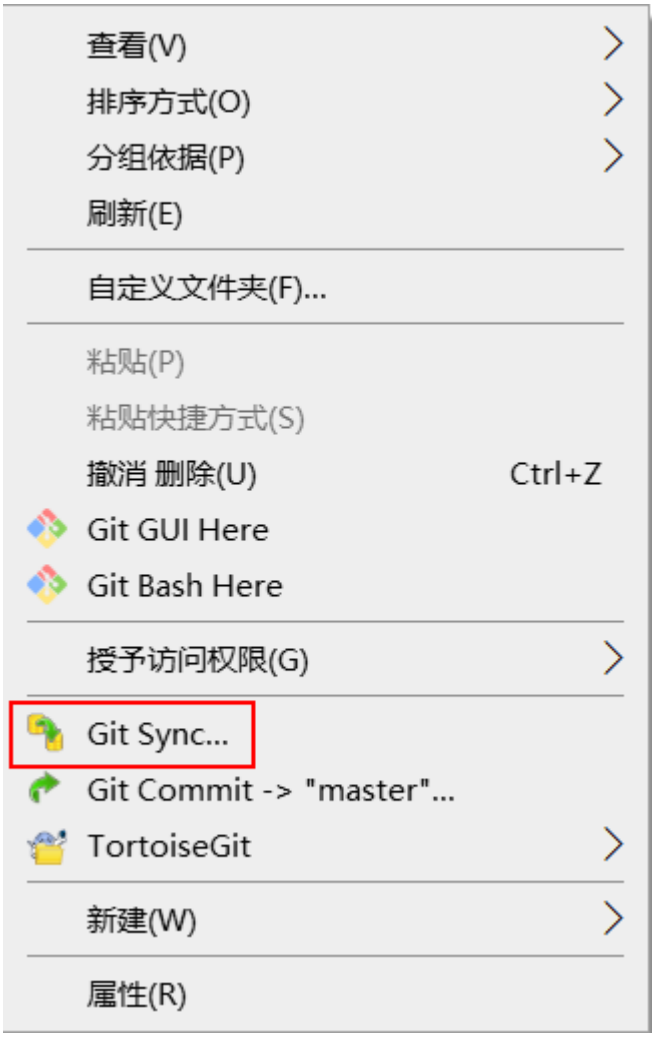
先将修改后的 helloworld.py 提交到本地仓库，再推送到远程仓库。



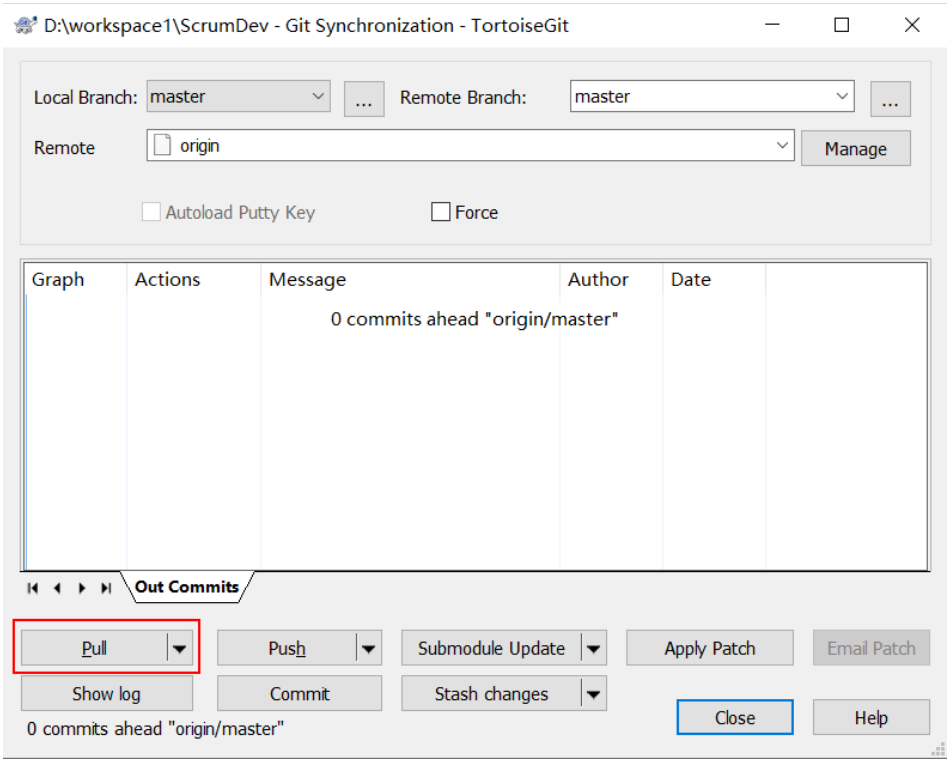
现在我们需要使用 TortoiseGit 将开发人员 B 的修改更新到本地 Git 仓库。先进入原先 Git 仓库所在的文件夹。



在文件夹空白处右键，点击“Git Sync”。



在弹出的界面中，点击“Pull”，即可将远程仓库的更新拉取下来。



打开 helloworld.py 文件，可以看到内容已经更新成开发人员 B 修改后的状态。

3 Gitflow 工作流程实践

3.1 实验背景

现在我们已经学习了本地 Git 仓库的基本操作，以及与 CodeHub 中远程仓库进行交互的一些操作。现实软件开发中 Git 的使用方式往往更加复杂。一个合适的 Git 工作流程，能有效提高项目管理水平和团队协同开发能力。Git 工作流程是代码管理的分支策略，它服务于项目流程管理和团队协同开发。

Gitflow 工作流程是一种 Git 工作流程，一般用于管理大型项目。本实验指导读者通过一个案例，实践 Gitflow 工作流流程。本课程您会学习：

- 什么是 Gitflow 工作流程
- 实践 Gitflow 工作流程

3.1.1 Gitflow 工作流程

Gitflow 是一种代码管理的分支策略，一般用于管理大型项目，它为不同的分支分配一个很明确的工作角色，并定义分支之间什么时候进行交互，Gitflow 工作流程如下图所示。

工作方式

- **master 分支：**

生产分支，最稳定的版本，一直是 ready to deploy 状态。不接受开发人员直接 commit，只接受从其他分支 merge 操作。在很多企业中，这个分支被默认开启分支保护，只有维护者可以操作。

- **hotfix 分支：**

从 master 分支拉取的临时修复分支，用于解决一线紧急 bug。bug 解决后需要合入 master 分支并打上新的版本号，这个修改也需要同时合入 develop 分支。

- **develop 分支：**

从 master 分支拉取的开发分支，用于功能集成。包含所有要发布到下一个 Release 的代码，用于开发集成、系统测试。

- **release 分支：**

临近既定的发布日，就从 develop 分支上拉取一个 release 分支，任何不在当前分支中的新功能都推到下个发布中。release 分支用于发布，所以从当前时间点之后新的功能不能再加到这个分支上，这个分支只做 Bug 修复、文档生成和其它面向发布的任务。当对外发布的工作都完成了，release 分支合并到 master 分支并分配一个版本号打好 Tag；另外，这些从 release 分支新做的修改要反向合并回 develop 分支。

- **feature 分支：**

开发者使用的特性分支，父分支是 develop 分支，当新功能完成时，合入 develop 分支。新功能提交从不直接与 master 分支交互。

3.2 实验介绍

3.2.1 实验内容

假设你正在世界五百强的 ICT 公司工作，按团队分工，你负责组织开发一个限流的功能。为实现该功能，你创建了一个 feature 分支，并推送到远程仓库，方便小组人员共同开发该功能。完成该功能后，你将功能合入 develop 分支，并创建一个 release 分支进行发布。发布完成后，你将 release 分支合入 master 分支和 develop 分支。发布后，你突然接到一个电话说当前发布的版本有一个 bug。为了解决这个问题，你基于 master 分支创建了一个 hotfix 分支，在这个分支修复了 bug，然后把改动的代码合入 master 分支以及 develop 分支。

3.2.2 实验步骤

本实验步骤如下：

1. 初始化一个代码仓库，以 master 分支为基础创建 develop 分支。作为你们团队的代码仓库。
2. 克隆仓库到本地，以 develop 分支为基础创建 featureA 分支，并推送到远程仓库。
3. 在 featureA 分支添加限流功能后，推送到远程仓库。
4. 将 featureA 分支合入 develop 分支。

5. 在本地更新 develop 分支，并以此为基础创建 release-v1 分支，并推送到远程仓库。

6. 在 release-v1 分支进行一些面向发布的工作，并推送到远程仓库。

7. release-v1 分支工作完成后，将其合入 master 分支，并为 master 分支打上标签。将 release-v1 分支也合入 develop 分支。

8. 更新本地 master 分支和 develop 分支，并基于 master 分支创建 hotfix 分支。

9. 在 hotfix 进行 bug 修改工作。完成后，将 hotfix 分支推送到远程仓库。

10. 将 hotfix 分支合入 master 分支，并打上标签。将 hotfix 分支合入 develop 分支。

3.3 实验操作

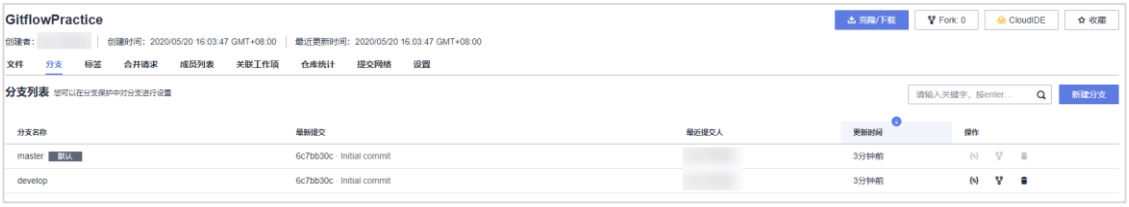
本实验通过 Git 命令行模式操作。

1. 在 CodeHub 创建一个新的代码仓库。并以 master 分支为基础创建 develop 分支。这个仓库就作为你们团队的远程仓库，你需要开发的限流功能也是在这个仓库上开发。

CodeHub 创建代码仓库在 2.3 节已有介绍，此处截图略。如下图，已在 CodeHub 创建了一个名为 GitflowPractice 的远程仓库。

以 master 为基础创建 develop 分支。

现在远程仓库已经有了 master 分支和 develop 分支。



用 Git 工作流程图来表示这一步的操作。

2. 你现在将远程仓库克隆到本地，并且以 develop 分支为基础创建 featureA 分支用来进行限流功能的开发，为了小组成员也能参与限流功能的开发，你将 featureA 分支推送到远程仓库。

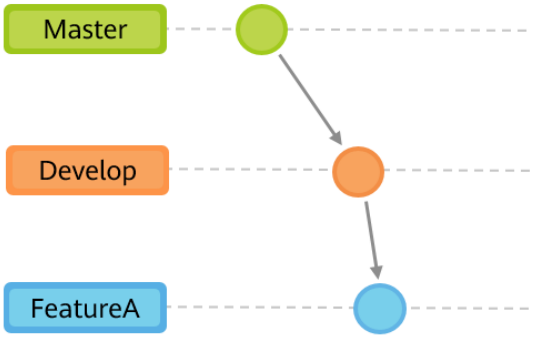
```
$ git clone git@codehub.devcloud.cn-north-4.huaweicloud.com:Scrum100005/GitflowPractice.git        #克隆远程仓库到本地
$ git checkout -b develop origin/develop          #本地创建 develop 分支并跟踪远程分支 origin/develop
$ git checkout -b featureA                        #以 develop 分支为基础创建 featureA 分支
$ git push -u origin featureA                     #将 featureA 分支推送到远程仓库
```

这时你的本地 Git 仓库已经拥有 develop、featureA、master 三个分支。

```
$ git branch                          #查看分支
  develop
* featureA
  master
```

在远程仓库也同样有这三个分支。

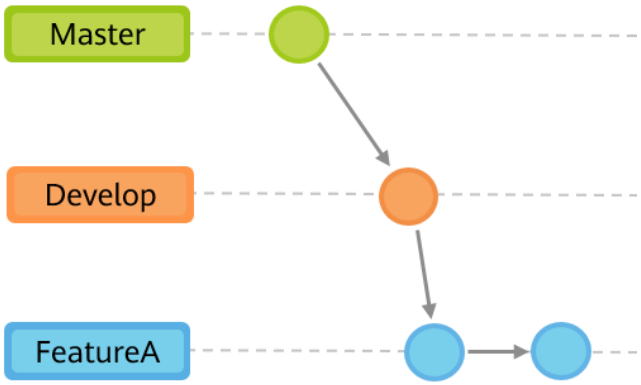
用 Git 工作流程图来表示这一步的操作。



3. 现在你在 featureA 分支添加限流功能，我们用新增一个 trafficControl.py 文件来表示限流功能的开发。开发完成后，你将更新推送到远程仓库。

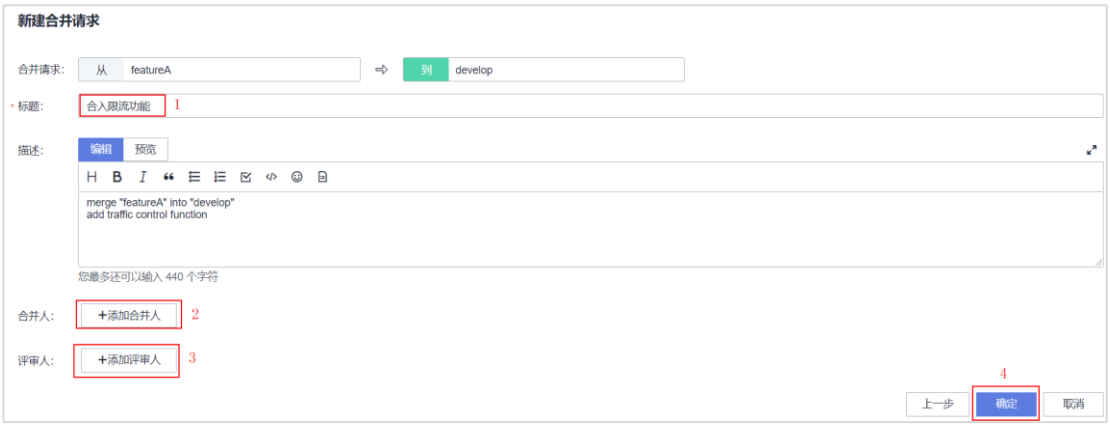
```
$ touch trafficControl.py                          #新增 trafficControl.py 文件
$ git add trafficControl.py                        #添加到暂存区
$ git commit -m "add traffic control function"     #提交到 Git 仓库，-m 选项表示指定提交版本库中的注释内容。
$ git push                                         #推送到远程仓库
```

用 Git 工作流程图来表示这一步的操作。

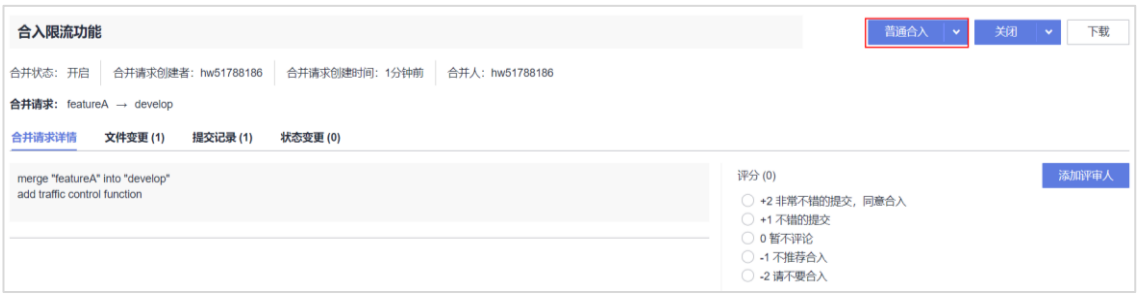


4. 上一步你已将限流功能开发完成了，你准备将代码合入 develop 分支，为了保证代码质量，你需要其他人开发人员进行 code review。所以你在 CodeHub 创建了合并申请，并指定其他开发人员进行评审。

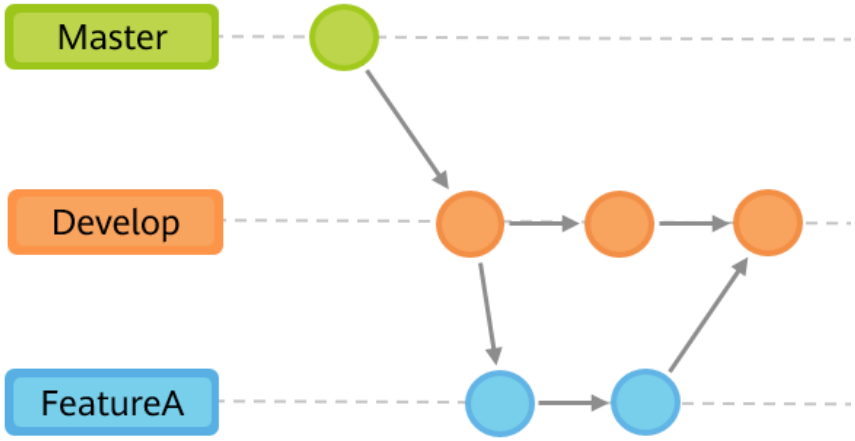
点击“下一步”后，在弹出的界面中输入标题字段。并添加合并人和评审人，最后点击“确定”。合并人是最终进行合并操作的人，评审人是进行 code review 的人。



之后评审人和合并人都会收到通知。本实验中评审人和合并人都选择了自己。评审人可以发布评审意见，开发人员根据评审意见继续修改。现在限流功能已经评审通过，合并人点击“普通合入”将其合并到了 develop 分支。



用 Git 工作流程图来表示这一步的操作（我们假设其他功能团队，在你合并 featureA 分之前，已经往 develop 分支合入了一些功能，所以 develop 分支会有中间的圆形）。



完成了限流功能的开发后，为了保持 Git 仓库分支结构的清晰，你将 featureA 分支删除。

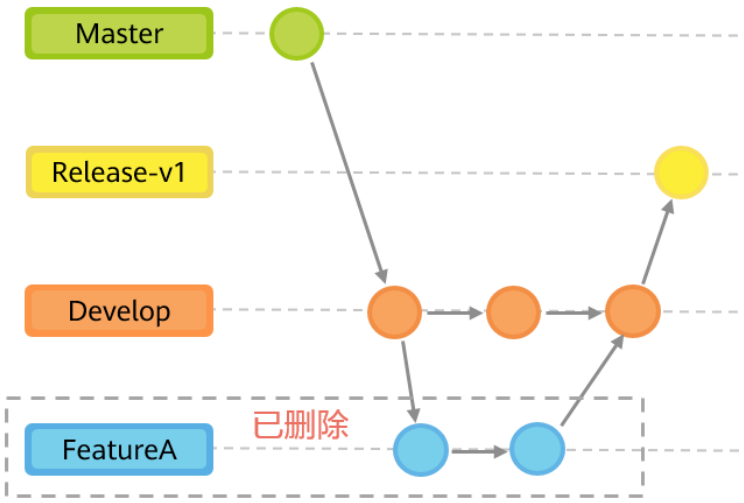
```
$ git switch develop              #在删除本地仓库 featureA 分支前，选切换到其他分支
$ git branch -d featureA          #删除本地仓库 featureA 分支
$ git push origin -d featureA     #删除远程仓库 featureA 分支
```

在远程仓库中可以看到，现在只有 master、develop 分支，featureA 分支已经删除。

5. 现在 develop 分支已经合入了不少功能，团队准备让你进行一次版本发布。你先在本地更新 develop 分支，然后以此为基础创建 release-v1 分支，并推送到远程仓库，方便团队成员协作进行一些发布的工作。

```
$ git switch develop                #切换到 develop 分支
$ git pull                          #更新 develop 分支
$ git branch release-v1             #以 develop 分支为基础创建 release-v1 分支
$ git switch release-v1             #切换到 release-v1 分支
$ git push -u origin release-v1     #将 release-v1 分支推送到远程仓库
```

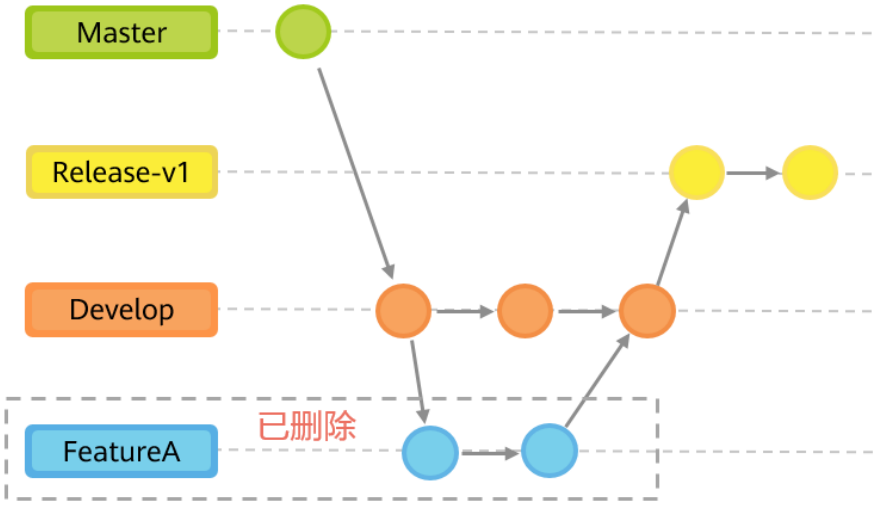
用 Git 工作流程图来表示这一步的操作。



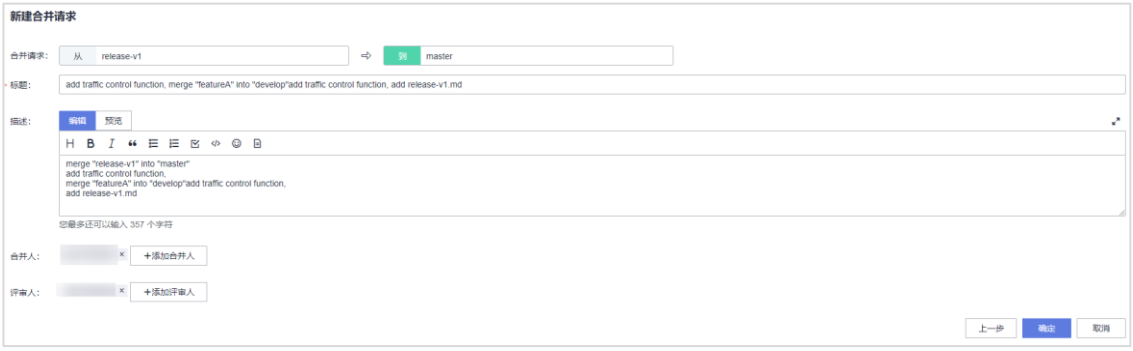
6. 现在你在 release-v1 分支进行一些面向发布的工作，我们用创建一个 release-v1.md 表示，并推送到远程仓库。

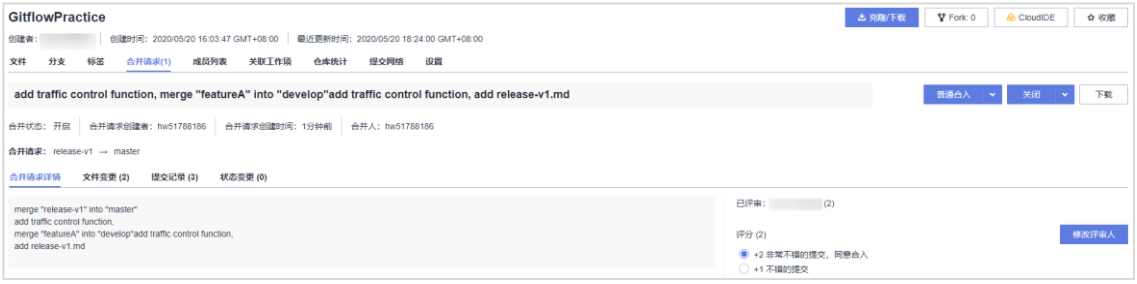
```
$ touch release-v1.md                          #创建 release-v1.md 文件
$ git add release-v1.md                        #添加到暂存区
$ git commit -m "add release-v1.md"            #提交到 Git 仓库
$ git push                                     #推送到远程仓库
```

用 Git 工作流程图来表示这一步的操作。

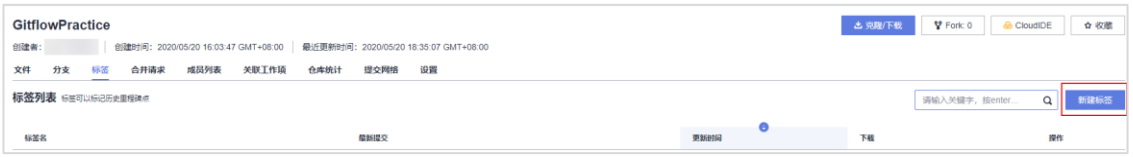


7. 现在你已将 release-v1 分支所涉及的发布工作完成，需要将其合入 master 分支，并为 master 分支打上标签。也需要将 release-v1 分支合入 develop 分支。这些步骤都在远程代码仓库操作。





将 release-v1 合入 master 后，为 master 分配一个版本号并打好标签。



在 master 分支上打好标签后，可以看到远程仓库中已经有了 v1 标签，标签其实就是 master 分支的一个快照，记录了这一时刻 master 分支的状态。

再将 release-v1 分支合并入 develop 分支，步骤与合并入 master 分支相同，截图略。

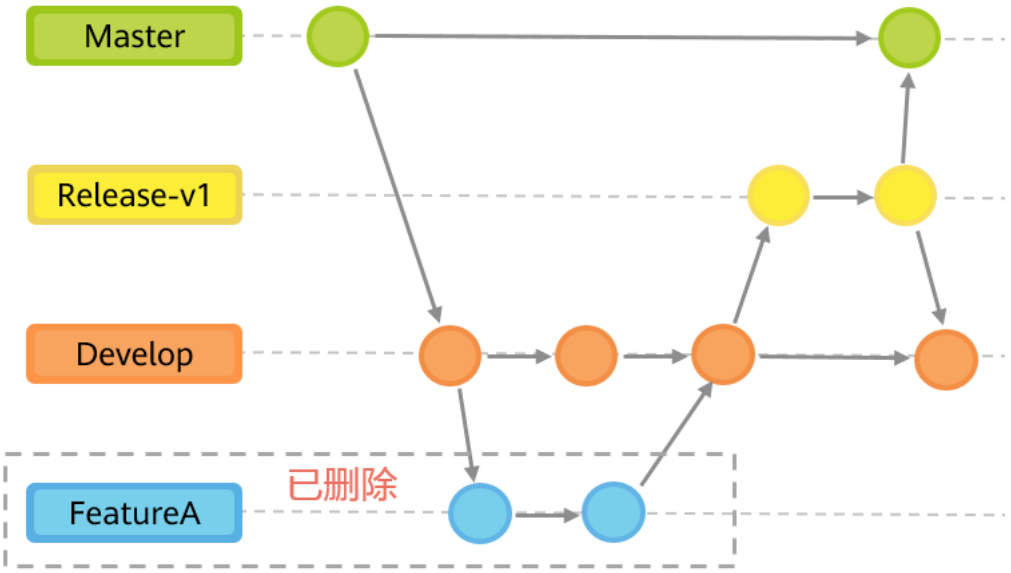
操作完成后，为保持 Git 仓库分支结构清晰，将 release-v1 分支删除。

```
$ git switch master                 #删除 release-v1 分支前，先切换到其他分支
$ git branch -d release-v1          #删除 release-v1 分支
$ git push origin -d release-v1     #删除远程仓库中的 release-v1 分支
```

删除 release-v1 分支后，可以看到在本地仓库和远程仓库都只有 master 和 develop 分支。

```
$ git branch                        #查看分支
  develop
* master
```

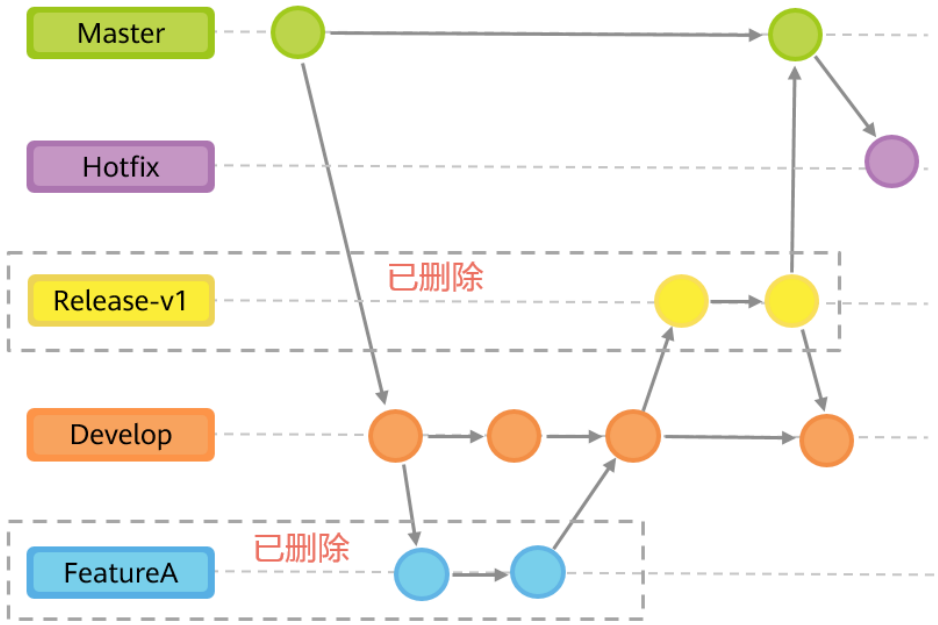
用 Git 工作流程图来表示这一步的操作。



8. 版本发布之后，你突然接到一个电话说当前发布的版本有一个 bug，为了解决这个问题。你先更新本地 master 分支和 develop 分支（因为远程仓库 master 和 develop 分支已有更新），并基于 master 分支创建 hotfix 分支，用于 bug 修复工作。

```
$ git switch develop                #切换到 develop 分支
$ git pull                          #更新 develop 分支
$ git switch master                 #切换到 master 分支
$ git pull                          #更新 master 分支
$ git branch hotfix                 #以 master 分支为基础创建 hotfix 分支
$ git switch hotfix                 #切换到 hotfix 分支
```

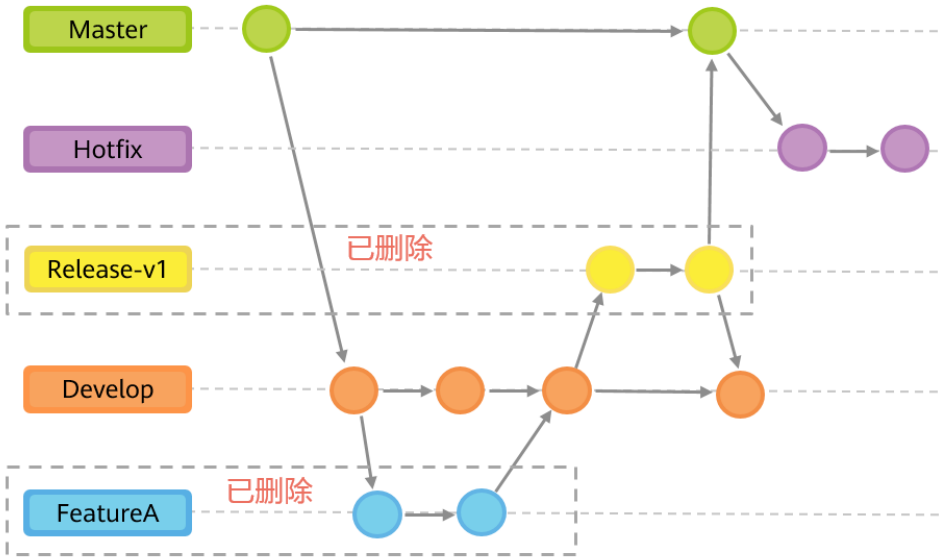
用 Git 工作流程图来表示这一步的操作。



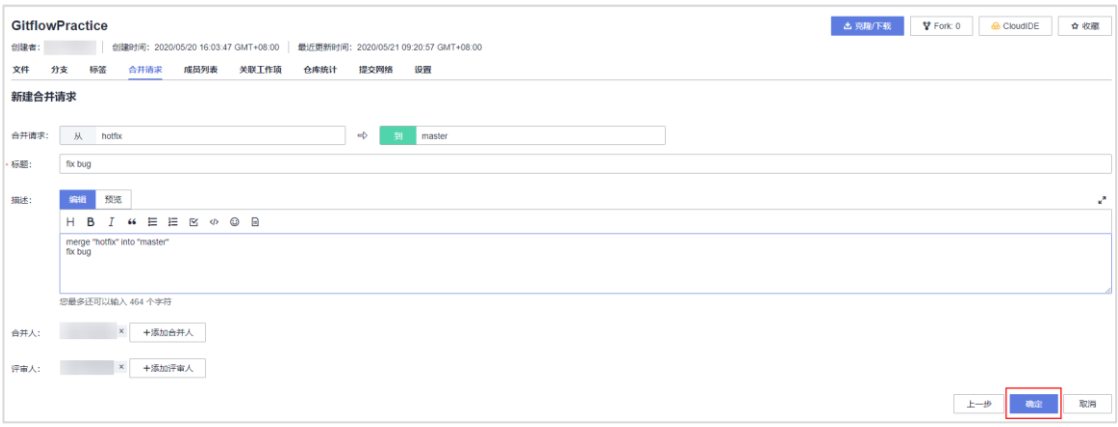
9. 在 hotfix 进行 bug 修改工作。我们用新增一个 bugfixed.pat 文件来代表 bug 修复工作。完成后，将 hotfix 分支推送到远程仓库。

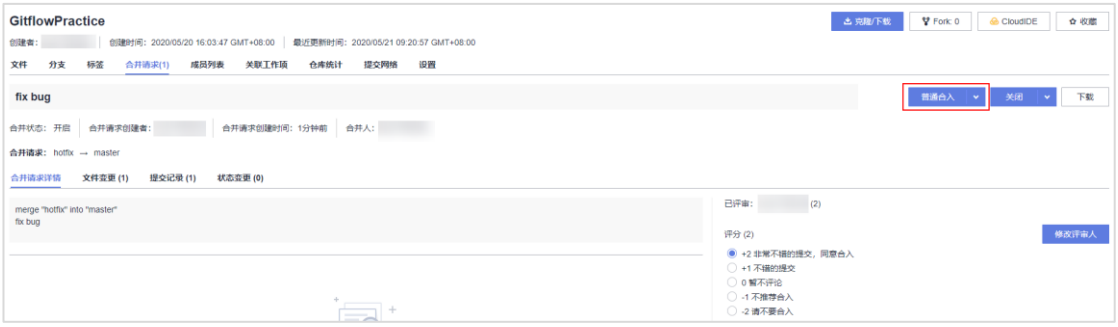
```
$ git switch hotfix
$ touch bugfixed.pat                          #新增一个 bugfixed.pat 文件
$ git add bugfixed.pat
$ git commit -m "fix bug"
$ git push -u origin hotfix                   #将 hotfix 分支推送到远程仓库
```

用 Git 工作流程图来表示这一步的操作。



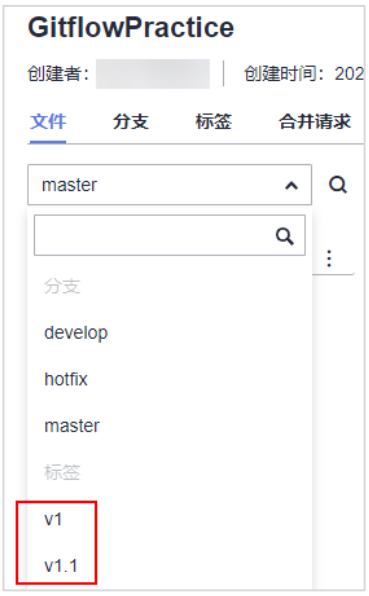
10. 现在 bug 修复工作已完成，你将 hotfix 分支合入 master 分支，并打上标签。之后，同样将 hotfix 分支合入 develop 分支。





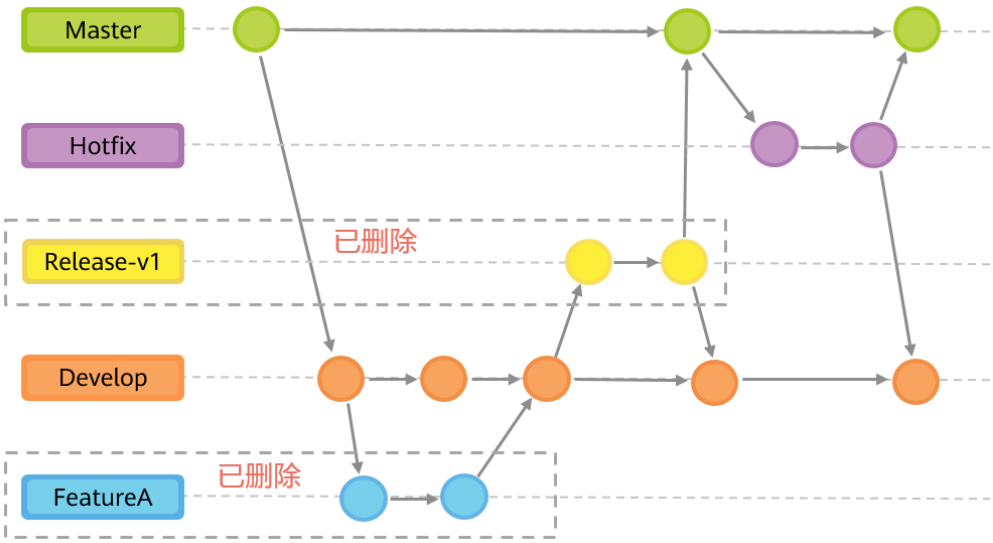
为 Master 打上新的标签 v1.1。

现在 master 分支已经有了 v1 和 v1.1 两个标签。



将 hotfix 分支合入 develop 分支，步骤与 hotfix 分支合入 master 分支相同，截图略。

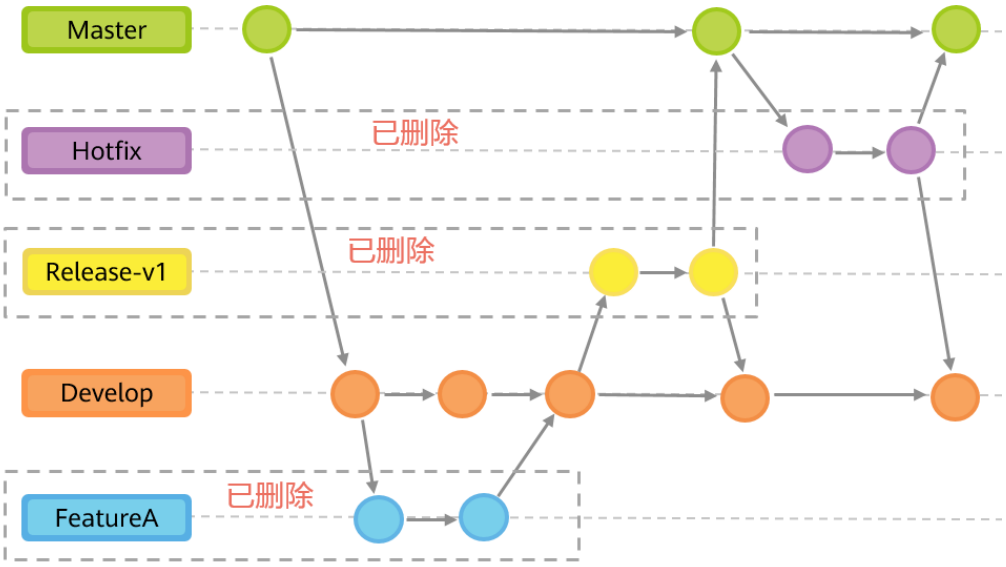
用 Git 工作流程图来表示这一步的操作。



最后，为了保持 Git 仓库分支结构的清晰，完成 bug 修复工作后，将 hotfix 分支删除。

```
$ git switch master                    #切换到其他分支
$ git branch -d hotfix                 #删除本地仓库 hotfix 分支
$ git push origin -d hotfix            #删除远程仓库 hotfix 分支
```

现在本地 Git 仓库和远程仓库就都只有 master 和 develop 分支了。最后的分支状态如下图。



至此，我们就完成了 Gitflow 工作流程的一个基本操作实践。

华为认证 HCIP 系列教程

HCIP-Datacom-Network Automation Developer

网络设备开放可编程

实验指导手册

版本:1.0

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： http://e.huawei.com

华为认证体系介绍

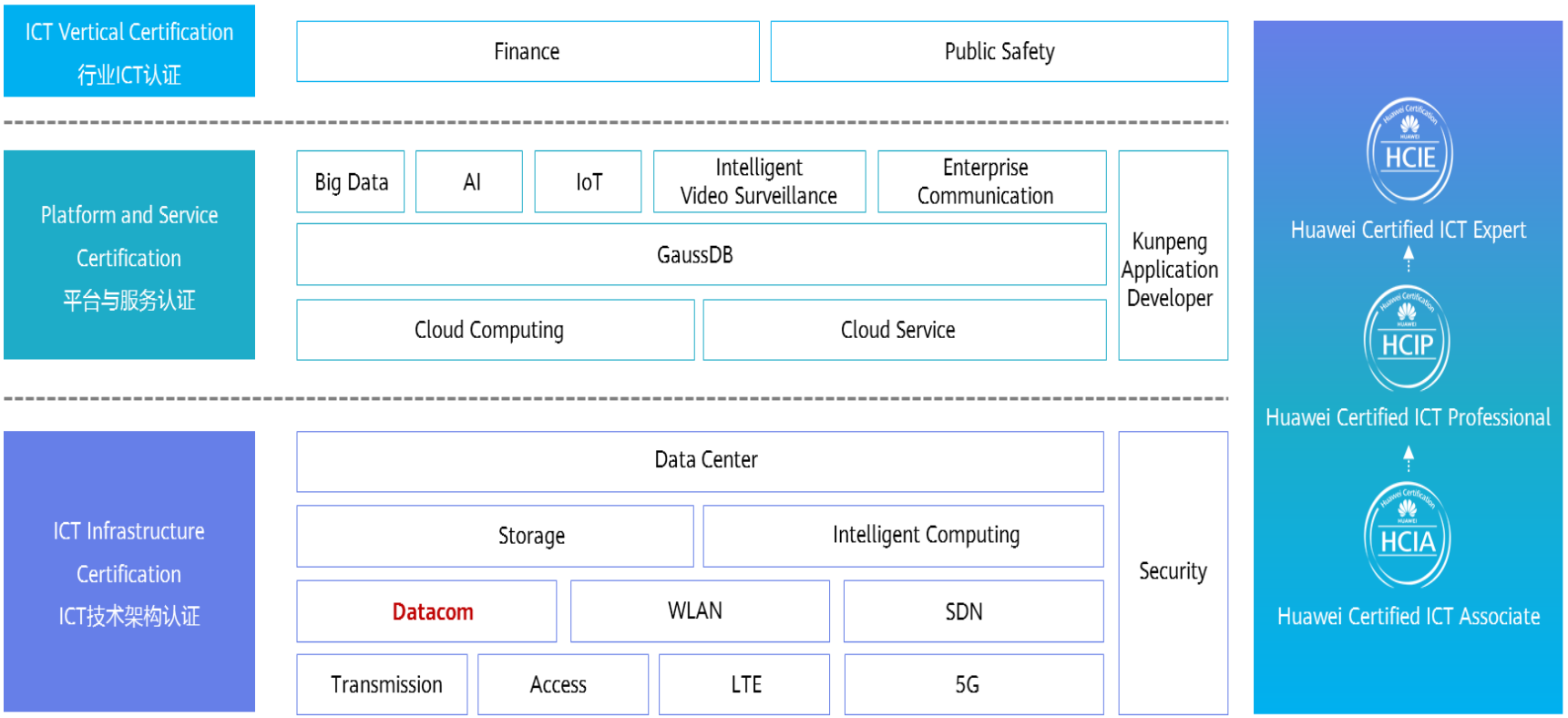
华为认证是华为公司基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，打造的ICT技术架构认证、平台与服务认证、行业ICT认证三类认证，是业界唯一覆盖ICT（Information and Communications Technology 信息通信技术）全技术领域的认证体系。

根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

HCIP-Datacom-Network Automation Developer定位于培养数通网络领域具备网络自动化开发专业知识和技能水平的高级工程师。通过HCIP-Datacom-Network Automation Developer认证将证明您能够胜任企业网络自动化开发工程师岗位，具备使用华为数通设备进行企业网络自动化部署、开发和运维的能力。

华为认证协助您打开行业之窗，开启改变之门，屹立在数通领域的潮头浪尖！

Huawei Certification



前 言

简介

本书为 HCIP-Datacom-Network Automation Developer 认证培训教程，适用于准备参加 HCIP-Datacom-Network Automation Developer 考试的学员，或者希望了解华为设备编程自动化知识和实践的读者。

内容描述

本实验指导书书共包含 7 个实验

- 实验一为 SSH 实验。
- 实验二为 SNMP 自动化配置实验。
- 实验三为 NETCONF 配置实验。
- 实验四为配置文件对比实验。
- 实验五为 gRPC 远程查询配置实验。
- 实验六为 Telemetry 配置实验。
- 实验七为 OPS 实验。
- 实验八为网络流量分析实验。

读者知识背景

本文档主要适用于进阶学习的网络自动化工程师。读者需具备以下知识和技能：

- Python 编程基础。
- 基本的网络知识，同时熟悉华为交换设备配置。

实验环境说明

本实验环境基于华为设备 VRP8，开发环境采用 Python 3.8，编译器为 Jupyter Notebook 或 Pycharm。

本实验设备可使用真实设备或者网络仿真平台。本手册将以 CE12800 系列设备为例，实现多种功能的 Python 自动化示例。

目录

1 SSH 实验

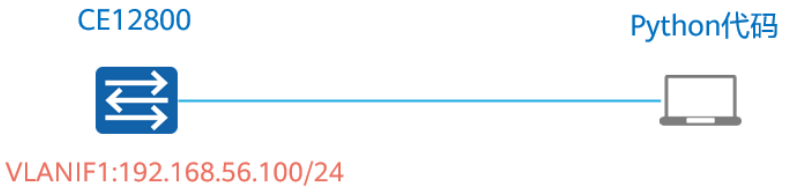
1.1 实验背景

某公司现有一台 CE12800 设备，管理 IP 地址为 192.168.56.100/24。现在需要编写自动化脚本，抓取设备当前配置文件及完成配置文件的上传和下载功能。

1.1.1 实验目标

本实验你将掌握 Paramiko 模块的常见方法，使用 SSH 登录指定设备实现配置自动化和 SFTP 安全文件传输。

1.1.2 实验环境准备



本实验需要准备一台网络设备和 Python 编译环境三层互通。本例中 Python 编译环境为本地 PC。

1.配置 CE12800。

```
<HUAWEI>system-view immediately
Enter system view, return user view with return command.
[HUAWEI]sysname CE1
[CE1]interface Vlanif 1
[CE1-Vlanif1]ip add 192.168.56.100 24
[CE1-Vlanif1]quit
```

2.验证可达性。

登录本地 CMD 窗口测试本机到 CE1 的连通性。

```
C:\Users\XXX>ping 192.168.56.100
正在 Ping 192.168.56.100 具有 32 字节的数据:
来自 192.168.56.100 的回复: 字节=32 时间=3ms TTL=255
来自 192.168.56.100 的回复: 字节=32 时间=3ms TTL=255
来自 192.168.56.100 的回复: 字节=32 时间=2ms TTL=255
```

```
来自 192.168.56.100 的回复: 字节=32 时间=4ms TTL=255
192.168.56.100 的 Ping 统计信息:
    数据包: 已发送 = 4，已接收 = 4，丢失 = 0 (0% 丢失)，
往返行程的估计时间(以毫秒为单位):
    最短 = 2ms，最长 = 4ms，平均 = 3ms
```

连通成功。

1.2 Paramiko SSH 登陆设备

Paramiko 是 Python 最为常用的 SSH 模块，它支持口令或者公钥两种方式，可以实现安全的远程命令执行、文件传输等功能。

本例以 RSA 用户认证方式介绍客户端使用 Python Paramiko SSH 登录服务器的配置过程。

1.2.1 配置思路

服务器端的配置：

1. 配置设备 STelnet：配置管理 IP，使能 STelent 功能，配置用户界面。
2. 配置用户：创建本地用户和 SSH 用户，配置服务类型和认证方式。
3. 配置公钥：添加客户端生成的公钥并分配给用户。

客户端的配置：

1. 创建密钥对：本地生成公钥和私钥。
2. 编写 Python 代码。
3. 结果验证。

1.2.2 具体配置

本地 Python 脚本使用 SSH 登录设备前，需要首先在设备上创建 SSH 账户和开启 Stelnet 功能。

步骤 1 使能服务器端的 STelnet 功能及配置 VTY 用户界面。

```
[SSH Server] stelnet server enable
[SSH Server] user-interface vty 0 4
[SSH Server-ui-vty0-4] authentication-mode aaa
[SSH Server-ui-vty0-4] protocol inbound ssh
[SSH Server-ui-vty0-4] user privilege level 3
[SSH Server-ui-vty0-4] quit
```

步骤 2 在服务器端创建本地用户 python，将用户加入管理员组，并配置用户服务方式。

```
[SSH Server] aaa
[SSH Server-aaa] local-user python password irreversible-cipher Huawei12#$
[SSH Server-aaa] local-user python user-group manage-ug
[SSH Server-aaa] local-user python service-type ssh
```

```
[SSH Server-aaa] quit
```

步骤 3 在服务器端创建 SSH 用户，并配置认证方式和服务方式。

```
[SSH Server] ssh user python
[SSH Server] ssh user python authentication-type rsa
[SSH Server] ssh user python service-type stelnet
```

步骤 4 客户端使用 Git Bash 创建 RSA 密钥对，并将公钥拷贝。

本步骤直接回车保持默认值。

```
exampleuser@exampleuser MINGW64 ~
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/exampleuser/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/exampleuser/.ssh/id_rsa
Your public key has been saved in /c/Users/exampleuser/.ssh/id_rsa.pub
```

显示公钥。

```
$ cat /c/Users/exampleuser/.ssh/id_rsa.pub
```

步骤 5 SSH 服务器添加公钥，并将公钥分配给用户。

```
[SSH Server] rsa peer-public-key rsa01 encoding-type openssh
[SSH Server-rsa-public-key] public-key-code begin
[SSH Server-rsa-public-key-rsa-key-code] ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQDwLRx8MmuNs500dRemhFHdDbBmxco8Bp+wyqwaGuHJZBCjy
FQV6AB+ezu5t0eWE3mw57IZfgmvR+MjBcliZv/x3l8oUMLcQKlKslYQDtvfUCZd+za1suXAPB/dyPKMhYPAzS
DA7K+xqCWlmU3q06vxHEPLMv4A5IX54rKtBnK92fWjl9ACU+ak0ZlHxbKwOFn1tr0GJBazcInEs9DKGwkTTq
Jdu9+5hI5NxXTSbM3an53805ZbCU18xPy57g7MZC89vbdsag/uvQmFkLJ3arts/Om2R7fhR92EU/SNPmVy+
qDEdwZEVdubdqJInW+8zzVkPGlnb2oH5hwH78Ksklbxb0fEfmGR0mS1ZAi3ZHUGcEEjuFZona3+5Z0Un2OP
xfXwvoljVDusbYcugJHo9Ssurz05GzVuamQZlcO2JYY6FhtLUAImtXGQ80MpTjB0lcprkAZCib8agYOtVQNTZ7
iB0g2EcBN9UTyMz7sh8RtrBDj445r+XPaDE8LmpDRKHMk=
[SSH Server-rsa-public-key-rsa-key-code] public-key-code end
[SSH Server-key-code] peer-public-key end
[SSH Server] ssh user python assign rsa-key rsa01
```

步骤 6 编写和运行 Python 代码，SSH 至服务器。

1.2.3 完整代码和运行结果

Python 脚本调用 paramiko 模块登录 CE1，执行 display current-configuration，并输出回显内容。

步骤 1 完整代码：

```
import paramiko
import time

ssh = paramiko.SSHClient()
```

```
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())

ssh.connect(hostname='192.168.56.100',port=22,username='python',key_filename=r'C:\Users\exampleuser\.ssh\id_rsa')

cli = ssh.invoke_shell()
cli.send('screen-length 0 temporary\n')
cli.send('display cu\n')
time.sleep(3)

dis_cu = cli.recv(999999).decode()
print(dis_cu)

ssh.close()
```

步骤 2 编译器执行：

步骤 3 输出结果：

```
Info: The max number of VTY users is 5, the number of current VTY users online is 1, and total number of terminal users online is 2.
      The current login time is 2020-05-12 11:04:56.
<SSH Server>screen-length 0 temporary
Info: The configuration takes effect on the current user terminal interface only.
<SSH Server>display cu
!Software Version V200R005C10SPC607B607
!Last configuration was updated at 2020-05-12 10:46:40+00:00
!Last configuration was saved at 2020-05-12 10:46:42+00:00
#
sysname SSH Server
#
device board 17 board-type CE-MPUB
device board 1 board-type CE-LPUE
#
rsa peer-public-key rsa01 encoding-type openssh
 public-key-code begin
  ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQDwLRx8MmuNs500dRemhFHdDbBmxco8Bp+wyqwaGuHJ
```

```
  ZBCjyFQV6AB+ezu5t0eWE3mw57IZfgmvR+MjBcliZv/x3l8oUMLcQKlKslYQDtvfUCZd+za1suXAPB/d
  yPKMhYPAzSDA7K+xqCWlmU3q06vxHEPLMv4A5IX54rKtBnK92fWjl9ACU+ak0ZlHxbKwOFn1tr0GJBaz
  cInEs9DKGwkTTqJdu9+5hI5NxXTSbM3an53805ZbCU18xPy57g7MZC89vbdsag/uvQmFkLJ3arts/Om2
  R7fhR92EU/SNPmVy+qDEdwZEVdubdqJInW+8zzVkPGlnb2oH5hwH78Ksklbxb0fEfmGR0mS1ZAi3ZHUG
  cEEjuFZona3+5Z0Un2OPxfXwvoljVDusbYcugJHo9Ssurz05GzVuamQZlcO2JYY6FhtLUAImtXGQ80Mp
  TjB0lcprkAZCib8agYOtVQNTZ7iB0g2EcBN9UTyMz7sh8RtrBDj445r+XPaDE8LmpDRKHMk= rsa-key

 public-key-code end
 peer-public-key end
#
aaa
 local-user python password irreversible-cipher
$1c$'&*wWaA2L3$iU"Y!^^Y}VJFkK=_E%)H({;pWU!Zr6]NO<LyLS,0$
 local-user python service-type ssh
 local-user python user-group manage-ug
 #
 authentication-scheme default
 #
 authorization-scheme default
 #
 accounting-scheme default
 #
 domain default
 #
 domain default_admin
#
interface Vlanif1
 ip address 192.168.56.100 255.255.255.0
#
interface MEth0/0/0
 undo shutdown
#
interface GE1/0/0
 undo shutdown
#
interface GE1/0/1
 undo shutdown
#
interface GE1/0/2
 shutdown
#
interface GE1/0/3
 shutdown
#
interface GE1/0/4
 shutdown
#
interface GE1/0/5
```

```
 shutdown
#
interface GE1/0/6
 shutdown
#
interface GE1/0/7
 shutdown
#
interface GE1/0/8
 shutdown
#
interface GE1/0/9
 shutdown
#
interface NULL0
#
stelnet server enable
ssh user python
ssh user python authentication-type rsa
ssh user python assign rsa-key rsa01
ssh user python service-type stelnet
ssh authorization-type default root
#
ssh server cipher aes256_gcm aes128_gcm aes256_ctr aes192_ctr aes128_ctr aes256_cbc aes128_cbc
3des_cbc
#
ssh server dh-exchange min-len 1024
#
ssh client cipher aes256_gcm aes128_gcm aes256_ctr aes192_ctr aes128_ctr aes256_cbc aes128_cbc
3des_cbc
#
user-interface con 0
#
user-interface vty 0 4
 authentication-mode aaa
 user privilege level 3
 protocol inbound ssh
#
vm-manager
#
return
<SSH Server>
```

1.2.4 代码解析

步骤 1 导入模块

```
import paramiko
```

```
import time
```

导入本段代码中需要使用的 paramiko 和 time 两个模块。如果没有安装此模块，可以 pip install paramiko 进行安装。

本章节主要介绍 Paramiko 作为客户端的常用类和方法，例如常用类 SSHClient 类及其涵盖的 AutoAddPolicy、connect、invoke_shell 和 close 等方法。更多 Paramiko 的方法请参考 http://docs.paramiko.org/。

Python 默认无间隔按顺序执行所有代码，在使用 paramiko 向交换机发送配置命令时候可能会遇到 SSH 响应不及时或者设备回显信息显示不全。此时，可以使用 time 模块下的 sleep 方法来人为暂停程序。

步骤 2 实例化 SSH 对象

使用 Paramiko SSHClient()实例化 SSH 对象。本例赋值给 ssh。

```
ssh = paramiko.SSHClient()
```

步骤 3 允许连接未知主机

即新建立 ssh 连接时不需要再输入 yes 或 no 进行确认。

```
ssh.set_missing_host_key_policy(paramiko.client.AutoAddPolicy())
```

步骤 4 建立 SSH 会话连接

目的 SSH 服务器为 192.168.56.100，用户名为 python, key_filename 指定客户端本地的私钥文件，以密钥方式进行用户认证。

```
ssh.connect(hostname='192.168.56.100',username='python',key_filename=r'C:\Users\exampleuser\.ssh\id_rsa')
```

步骤 5 打开交互式会话

SSH 到设备后，使用 Python 脚本向 SSH Server 输入执行命令。

调用 invoke_shell()赋值给 cli。invoke_shell()作用是打开一个交互的 shell 会话。该会话为一个逻辑通道 channel，建立在 SSH 会话连接上。

```
cli = ssh.invoke_shell()
```

使用 send()继续输入命令行。

Screen-length 0 temporary 意思是取消分屏，一次性输出所有回显命令。display cu 为 display current-configuration 缩写，作用是显示当前配置。

time.sleep()设置等待时间(秒)。

```
cli.send('screen-length 0 temporary\n')
cli.send('display cu\n')
time.sleep(3)
```

步骤 6 抓取 channel 回显信息

invoke_shell()已经创建了一个 channel 逻辑通道。此前所有的输入输出的过程信息都在此 channel 中。我们可以获取这个 channel 中所有信息，显示到 Python 编译器。

```
dis_cu = cli.recv(999999).decode()
print (dis_cu)
```

调用 cli.recv()，然后使用 decode()进行对其解码，最后赋值给 dis_cu。

recv(999999)作用是接收 channel 中的数据，数据最大量为 999999 bytes。

decode()方法作用是以指定的编码格式解码 bytes 对象，默认编码格式为 utf-8。

解码作用是方便阅读，结果将换行呈现到界面：

```
Info: The max number of VTY users is 5, the number of current VTY users online is 2, and total number of terminal users online is 3.
      The current login time is 2019-11-05 16:43:26.
      The last login time is 2019-11-05 16:43:09 from 192.168.56.1 through SSH.
<CE1>screen-length 0 temporary
Info: The configuration takes effect on the current user terminal interface only.
<CE1>display cu
!Software Version V200R005C10SPC607B607
!Last configuration was updated at 2019-11-05 14:33:28+00:00
#
```

否则显示如下：

```
nInfo: The max number of VTY users is 5, the number of current VTY users online is 2, and total number of terminal users online is 3.\r\n      The current login time is 2019-11-05 16:45:34.\r\n      The last login time is 2019-11-05 16:43:26 from 192.168.56.1 through SSH.\r\n<CE1>screen-length 0 temporary\r\nInfo: The configuration takes effect on the current user terminal interface only.\r\n<CE1>display cu\r\n!Software Version V200R005C10SPC607B607\r\n!Last configuration was updated at 2019-11-05 14:33:28+00:00\r\n#
```

步骤 7 关闭会话连接

```
ssh.close()
```

调用 close()关闭当前会话连接。设备 vty 连接数量有限，在执行完脚本后需要关闭此 SSH 会话。

1.3 Paramiko SFTP 文件传输

SFTP（SSH File Transfer Protocol）是一个安全的文件传输协议，建立在 SSH 协议的基础之上。

本例以 RSA 用户认证方式介绍客户端使用 Python Paramiko SFTP 上传和下载文件的配置过程。

1.3.1 配置思路

服务器端的配置：

1. 配置设备 SFTP：配置管理 IP，使能 SFTP

2. 配置用户：创建 SSH 用户，配置服务类型和认证方式和 sftp 路径
3. 配置公钥：添加客户端生成的公钥并分配给用户
4. 结果验证：查看上传的文件

客户端的配置：

1. 创建密钥对：本地生成公钥和私钥
2. 编写 Python 代码
3. 结果验证：查看下载的文件

1.3.2 具体配置

本地 Python 脚本使用 SFTP 文件传输前，需要首先在设备上创建 SFTP 账户和开启 SFTP 功能。

步骤 1 使能 SFTP 服务器功能。

```
[SFTP Server] sftp server enable
```

步骤 2 创建用户 python 并配置认证类型和服务类型。

```
[SFTP Server] ssh user python
[SFTP Server] ssh user python authentication-type rsa
[SFTP Server] ssh user python service-type sftp
[SFTP Server] ssh user python sftp-directory cfcard:
[SFTP Server] ssh authorization-type default root
```

步骤 3 客户端使用 Git Bash 创建 RSA 密钥对，并将公钥拷贝。

```
exampleuser@exampleuser MINGW64 ~
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/exampleuser/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/exampleuser/.ssh/id_rsa
Your public key has been saved in /c/Users/exampleuser/.ssh/id_rsa.pub
```

将显示的公钥拷贝。

```
$ cat /c/Users/exampleuser/.ssh/id_rsa.pub
```

步骤 4 服务器端添加已复制的公钥，并将公钥分配给用户。

```
[SFTP Server] rsa peer-public-key rsa01 encoding-type openssh
[SFTP Server-rsa-public-key] public-key-code begin
[SFTP Server-rsa-public-key-rsa-key-code] ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQDwLRx8MmuNs500dRemhFHdDbBmxco8Bp+wyqwaGuHJZBCjy
FQV6AB+ezu5t0eWE3mw57IZfgmvR+MjBcliZv/x3l8oUMLcQKlKslYQDtvfUCZd+za1suXAPB/dyPKMhYPAzS
DA7K+xqCWlmU3q06vxHEPLMv4A5IX54rKtBnK92fWjl9ACU+ak0ZlHxbKwOFn1tr0GJBazcInEs9DKGwkTTq
Jdu9+5hI5NxXTSbM3an53805ZbCU18xPy57g7MZC89vbdsag/uvQmFkLJ3arts/Om2R7fhR92EU/SNPmVy+
qDEdwZEVdubdqJInW+8zzVkPGlnb2oH5hwH78Ksklbxb0fEfmGR0mS1ZAi3ZHUGcEEjuFZona3+5Z0Un2OP
```

```
xfXwvoljVDusbYcugJHo9Ssurz05GzVuamQZlcO2JYY6FhtLUAImtXGQ80MpTjB0lcprkAZCib8agYOtVQNTZ7
iB0g2EcBN9UTyMz7sh8RtrBDj445r+XPaDE8LmpDRKHMk=
[SFTP Server-rsa-public-key-rsa-key-code] public-key-code end
[SFTP Server-key-code] peer-public-key end
[SFTP Server] ssh user python assign rsa-key rsa01
```

步骤 5 客户端编写和运行 Python 代码，SFTP 至服务器进行文件上传和下载。

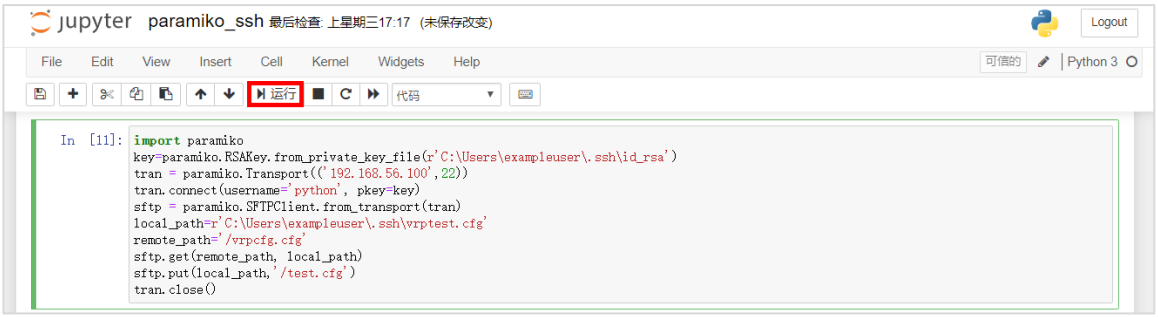
1.3.3 完整代码和运行结果

Python 脚本调用 paramiko 模块登录 CE1，下载设备配置文件“vrpcfg.cfg”，然后上传配置文件“test.cfg”。

步骤 1 完整代码：

```
import paramiko
key=paramiko.RSAKey.from_private_key_file(r'C:\Users\exampleuser\.ssh\id_rsa')
tran = paramiko.Transport(('192.168.56.100', 22))
tran.connect(username='python', pkey=key)
sftp = paramiko.SFTPClient.from_transport(tran)
local_path=r'C:\Users\exampleuser\.ssh\vrptest.cfg'
remote_path= '/vrpcfg.cfg'
sftp.get(remote_path, local_path)
sftp.put(local_path,'/test.cfg')
tran.close()
```

步骤 2 编译器执行：



步骤 3 结果验证：

客户端本地指定路径下（本例中：C:\Users\exampleuser\.ssh\vrptest.cfg）获得下载的文件 vrptest.cfg

服务器端输入 dir 命令查看，成功获得上传的指定文件 test.cfg。

```
<SFTP Server>dir
Directory of cfcard:/

  Idx  Attr      Size(Byte)  Date        Time        FileName
    0  dr-x              -  May 13 2020 14:45:05    $_checkpoint
    1  dr-x              -  Apr 28 2020 20:20:09    $_install_mod
    2  dr-x              -  Apr 28 2020 20:20:37    $_license
    3  dr-x              -  May 13 2020 14:44:23    $_security_info
    4  dr-x              -  May 13 2020 14:44:22    $_system
    5  -rw-              0  May 13 2020 14:43:48    CE12800
    6  -rw-            104  Apr 28 2020 20:20:09    VRPV200R005C10SP
_ce12800.cc
    7  -rw-            251  May 13 2020 14:43:48    device.sys
    8  drwx              -  May 06 2020 17:22:02    test
    9  -rw-          2,172  May 13 2020 17:17:01    test.cfg
   10  -rw-          2,172  May 13 2020 14:43:48    vrpcfg.cfg
```

1.3.4 代码解析

步骤 1 导入模块

```
import paramiko
```

导入本段代码中需要使用的 paramiko 模块。如果没有安装此模块，可以 pip install paramiko 进行安装。

本次 SFTP 涉及这三个类及其相关方法：Transport 类，Key handling 类， SFTPClient 类。

Transport 类用于实例化会话通道，建立会话连接。

Key handling 类用于实例化密钥对象。

SFTPClient 类用于创建 SFTP 会话连接并执行远程文件操作。

步骤 2 创建 RSA 密钥对象

读取客户端本地 RSA 私钥文件，赋值给 key。

```
key=paramiko.RSAKey.from_private_key_file(r'C:\Users\exampleuser\.ssh\id_rsa')
```

步骤 3 实例化会话通道，目的 SSH 服务器为 192.168.56.100，端口为 22

```
tran = paramiko.Transport(('192.168.56.100', 22))
```

步骤 4 建立 SSH 会话连接，用户名为 python, pkey 指定密钥对象，以密钥方式进行用户认证

```
tran.connect(username='python', pkey=key)
```

步骤 5 从打开的会话连接创建 SFTP 通道，赋值给 sftp

```
sftp = paramiko.SFTPClient.from_transport(tran)
```

步骤 6 设置本地路径和远端路径

```
local_path=r'C:\Users\exampleuser\.ssh\vrptest.cfg'
remote_path='/vrpcfg.cfg'
```

将“vrpcfg.cfg”重命名为 vrptest.cfg。

步骤 7 进行下载操作

```
sftp.get(remote_path, local_path)
```

步骤 8 进行上传操作

```
sftp.put(local_path,'/test.cfg')
```

步骤 9 关闭会话连接

```
tran.close()
```

1.4 思考题

如果我们需要使用 SSH 登陆多台设备查看配置如何实现?

2 SNMP 自动化配置实验

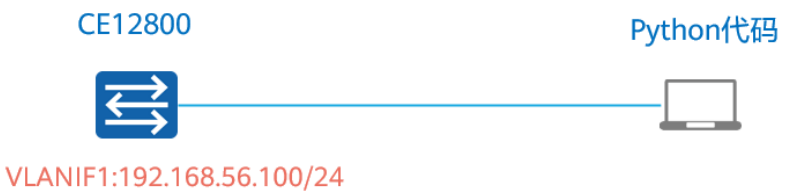
2.1 实验背景

某公司现有一台 CE12800 设备，管理 IP 地址为 192.168.56.100/24，可以 SSH 远程登陆。现有 SNMP 配置脚本 snmp.txt 文件。需要编写自动化脚本，首先进行设备 SNMP 配置，然后收集设备 SNMP 信息。

2.1.1 实验目标

- 掌握 open 对文件读写操作的方法
- 掌握 paramiko 模块作为 SSH 客户端常用方法
- 掌握 pysnmp 收集设备 SNMP 信息的方法

2.1.2 实验环境准备



本实验需要准备一台网络设备和 Python 编译环境三层互通。

1.配置 CE12800。

```
<HUAWEI>system-view immediately
Enter system view, return user view with return command.
[HUAWEI]sysname CE1
[CE1]interface Vlanif 1
[CE1-Vlanif1]ip add 192.168.56.100 24
[CE1-Vlanif1]quit
```

2.验证可达性。

登录本地 CMD 窗口测试本机到 CE1 的连通性。

```
C:\Users\XXX>ping 192.168.56.100
正在 Ping 192.168.56.100 具有 32 字节的数据:
来自 192.168.56.100 的回复: 字节=32 时间=3ms TTL=255
```

```
来自 192.168.56.100 的回复: 字节=32 时间=3ms TTL=255
来自 192.168.56.100 的回复: 字节=32 时间=2ms TTL=255
来自 192.168.56.100 的回复: 字节=32 时间=4ms TTL=255
192.168.56.100 的 Ping 统计信息:
    数据包: 已发送 = 4，已接收 = 4，丢失 = 0 (0% 丢失)，
往返行程的估计时间(以毫秒为单位):
    最短 = 2ms，最长 = 4ms，平均 = 3ms
```

连通成功。

3.设备 CE1 完成 SSH 配置，用户名 python，密码 Huawei12#$。（参考 1.2.2）

4.准备 SNMP 脚本，snmp.txt 文件。

SNMPv3 配置脚本中 SNMP 用户名 admin，密码 Huawei@123。完整脚本如下：

```
snmp-agent usm-user v3 admin group dc-admin
snmp-agen usm v3 admin au sha
Huawei@123
Huawei@123
snmp-a usm-user v3 admin pr aes128
Huawei@123
Huawei@123
snmp-agent trap source ME0/0/0
snmp-agent mib-view included nt iso
snmp-agent mib-view included rd iso
snmp-agent mib-view included wt iso
snmp-agent mib-view included iso-view iso
snmp-agent group v3 dc-admin privacy read-view rd write-view wt notify-view nt
```

2.2 配置思路

1. 完成实验环境准备。
2. 调用 paramiko 登陆设备。
3. 调用 open 打开本地文件 snmp.txt，结合 paramiko 配置设备。
4. 调用 pysnmp 通过设备 SNMP 信息获取主机名。

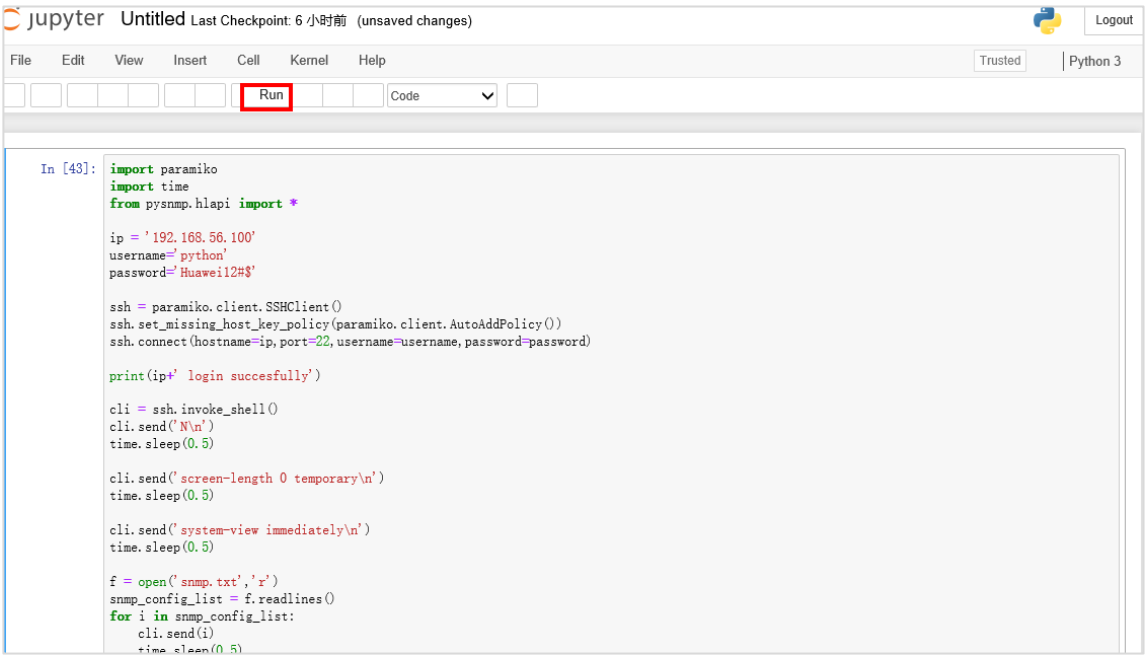
2.3 完整代码和运行结果

步骤 1 完整代码：

```
import paramiko
import time
from pysnmp.hlapi import *

#交换机信息
```

```
ip = '192.168.56.100'
username='python'
password='Huawei12#$'

#SSH 登陆设备
ssh = paramiko.client.SSHClient()
ssh.set_missing_host_key_policy(paramiko.client.AutoAddPolicy())
ssh.connect(hostname=ip,port=22,username=username,password=password)
print(ip+' login succesfully')

#打开一个 channel，输入配置
cli = ssh.invoke_shell()
cli.send('N\n')
time.sleep(0.5)
cli.send('screen-length 0 temporary\n')
time.sleep(0.5)

#进入系统视图
cli.send('system-view immediately\n')
time.sleep(0.5)

#逐行读取本地同一个文件夹里的 snmp.txt，写入 SSH 通道
f = open('snmp.txt','r')
snmp_config_list = f.readlines()
for i in snmp_config_list:
    cli.send(i)
    time.sleep(0.5)

#建立 SNMP 的通道
UdpTransportTarget((ip,161))
g = getCmd(SnmpEngine(),

#获取设备的主机名
UsmUserData('admin','Huawei@123','Huawei@123',authProtocol=usmHMACSHAAuthProtocol,privProtoc
ol=usmAesCfb128Protocol),
                UdpTransportTarget((ip, 161)),
                ContextData(),
                ObjectType(ObjectIdentity('SNMPv2-MIB','sysName',0)))
errorIndication, errorStatus, errorIndex, varBinds =next(g)
for i in varBinds:
    print (i)
    print (str(i).split('=')[1].strip())

dis_this = cli.recv(999999).decode()   #查看脚本交互过程
print (dis_this)
#关闭会话
ssh.close()
```

步骤 2 编译器执行：



步骤 3 输出结果：

```
192.168.56.100 login succesfully
SNMPv2-MIB::sysName.0 = CE1
CE1

Warning: The initial password poses security risks.
The password needs to be changed. Change now? [Y/N]:N

Info: The max number of VTY users is 5, the number of current VTY users online is 1, and total number of terminal users online is 2.
        The current login time is 2019-11-05 19:43:51.
        The last login time is 2019-11-05 19:43:07 from 192.168.56.1 through SSH.
<CE1>screen-length 0 temporary
Info: The configuration takes effect on the current user terminal interface only.
<CE1>system-view immediately
Enter system view, return user view with return command.
[CE1]snmp-agent usm-user v3 admin group dc-admin
[CE1]snmp-agen usm v3 admin authentication-mode sha
Please configure the authentication password (8-255)
Enter Password:
Confirm Password:
Warning: The privacy and authentication passwords are the same, which is insecure. It is recommended that the privacy and authentication passwords be different.
[CE1]snmp-a usm-user v3 admin privacy-mode aes128
Please configure the privacy password (8-255)
Enter Password:
Confirm Password:
```

```
Warning: The privacy and authentication passwords are the same, which is insecure. It is recommended
that the privacy and authentication passwords be different.
[CE1]snmp-agent trap source  Vlanif 1
[CE1]snmp-agent mib-view included nt iso
[CE1]snmp-agent mib-view included rd iso
[CE1]snmp-agent mib-view included wt iso
[CE1]snmp-agent mib-view included iso-view iso
[CE1]snmp-agent group v3 dc-admin privacy read-view rd write-view wt notify-view  nt
[CE1]
```

2.4 代码解析

步骤 1 导入模块

```
import paramiko
import time
from pysnmp.hlapi import *
```

导入本脚本使用的所有模块。如果没有安装 pysnmp，你可以使用 pip install pysnmp 安装。

pysnmp 用于获取 snmp 信息。本章节简单介绍同步 SNMP 的 Get 方法以获取设备信息，更多 pysnmp 信息请参考 http://snmplabs.com/pysnmp/。

步骤 2 登陆设备

```
#交换机信息
ip = '192.168.56.100'
username='python'
password='Huawei12#$'
```

创建变量 ip，username，password 用于表示 SSH 主机、用户名和密码。

```
#SSH 登陆设备
ssh = paramiko.client.SSHClient()
ssh.set_missing_host_key_policy(paramiko.client.AutoAddPolicy())
ssh.connect(hostname=ip,port=22,username=username,password=password)
print(ip+' login succesfully')
```

调用 paramiko 登陆设备，登陆成功返回主机 IP 地址 login successfully:

```
192.168.56.100 login succesfully
```

步骤 3 打开 SSH 通道

打开 SSH 通道，进行基础配置。

```
#打开一个 channel，输入配置
cli = ssh.invoke_shell()
cli.send('N\n')
time.sleep(0.5)
cli.send('screen-length 0 temporary\n')
```

```
time.sleep(0.5)

#进入系统视图
cli.send('system-view immediately\n')
time.sleep(0.5)
```

调用 invoke_shell()打开 SSH 通道。执行取消回显信息分屏和进入系统视图命令。执行命令间隔为 0.5 秒。设备 CLI 交互过程如下：

```
<CE1>screen-length 0 temporary
Info: The configuration takes effect on the current user terminal interface only.
<CE1>system-view immediately
Enter system view, return user view with return command.
```

步骤 4 配置设备 SNMP

配置设备 SNMP 信息。SNMP 用户名 admin，密码 Huawei@123，认证模式 SHA，privacy mode AES128。

```
#逐行读取本地同一个文件夹里的 snmp.txt，写入 SSH 通道
f = open('snmp.txt','r')
snmp_config_list = f.readlines()
for i in snmp_config_list:
    cli.send(i)
    time.sleep(0.5)
```

调用 Python 内置的 open 函数，读取 snmp.txt 信息形成列表赋值给 snmp_config_list。

使用 for 循环读取每行命令，写入 SSH 通道。

CLI 交互过程如下：

```
[CE1]snmp-agent usm-user v3 admin group dc-admin
[CE1]snmp-agen usm v3 admin authentication-mode sha
Please configure the authentication password (8-255)
Enter Password:
Confirm Password:
Warning: The privacy and authentication passwords are the same, which is insecure. It is recommended
that the privacy and authentication passwords be different.
[CE1]snmp-a usm-user v3 admin privacy-mode aes128
Please configure the privacy password (8-255)
Enter Password:
Confirm Password:
Warning: The privacy and authentication passwords are the same, which is insecure. It is recommended
that the privacy and authentication passwords be different.
[CE1]snmp-agent trap source Vlanif 1
[CE1]snmp-agent mib-view included nt iso
[CE1]snmp-agent mib-view included rd iso
[CE1]snmp-agent mib-view included wt iso
[CE1]snmp-agent mib-view included iso-view iso
[CE1]snmp-agent group v3 dc-admin privacy read-view rd write-view wt notify-view    nt
[CE1]
```

步骤 5 建立和设备的 SNMP 连接

```
#建立 SNMP 的通道
UdpTransportTarget((ip,161))
```

调用 pysnmp 里的 UdpTransportTarget()，参数包括目的 IP 和端口号。

更多参数细节请参考 http://snmplabs.com/pysnmp/docs/api-reference.html#synchronous-snmp。

步骤 6 SNMP GET 请求和回复

通过 pysnmp 的实现 SNMP 的 GET 操作获取设备信息。

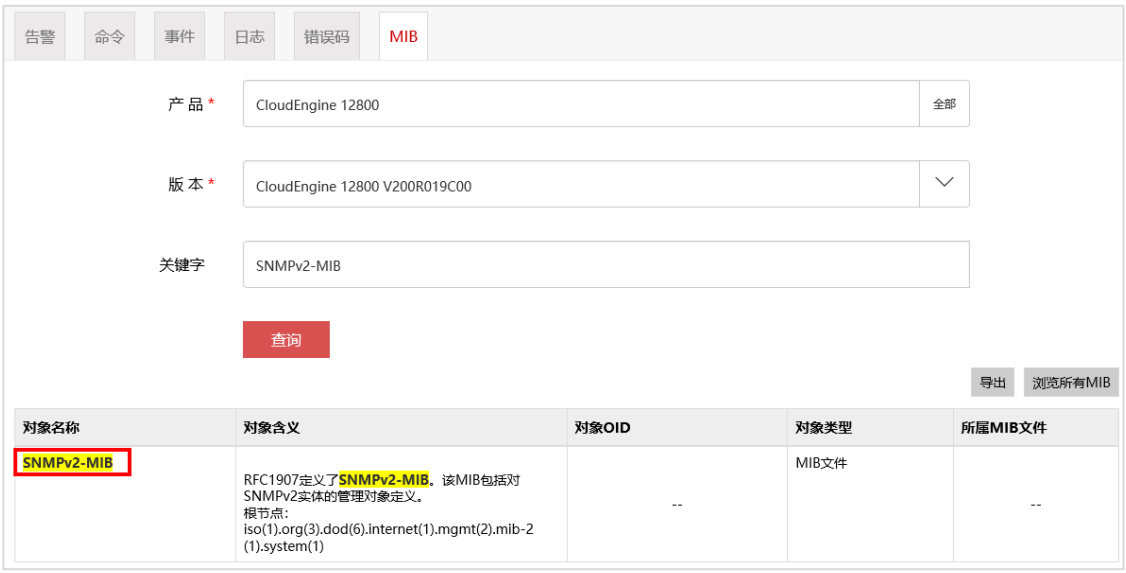
```
g = getCmd(SnmpEngine(),
UsmUserData('admin','Huawei@123','Huawei@123',authProtocol=usmHMACSHAAuthProtocol,privProtocol=usmAesCfb128Protocol),
            UdpTransportTarget((ip, 161)),
            ContextData(),
            ObjectType(ObjectIdentity('SNMPv2-MIB','sysName',0)))
```

调用 getCmd()实现 SNMP 的 GET 操作，赋值给 g。

- UsmUserData 为 SNMP 用户信息，包括 SNMP 用户名、密码、加密模式和认证模式。请和设备的 SNMP 配置保持一致。
- UdpTransportTarget 为传输层信息。
- ContextData 用于异步模式下，这里保持为空。
- ObjectType 为查询的设备 MIB 对象。可以使用对象名称，也可以使用 OID。这里使用的对象名称。

华为设备的 MIB 信息可以通过 http://support.huawei.com/onlinetoolsweb/infoM/index.do?domain=1&lang=zh&topicType=mib 查询。

当前查询的 MIB 对象文件名称为 SNMPv2-MIB。



具体对象为 sysName。


```
errorIndication, errorStatus, errorIndex, varBinds = next(g)
```

创建变量 errorIndication, errorStatus, errorIndex, varBinds 用于获取 next(g)的返回信息。

返回信息详细信息在官方文档中定义 http://snmplabs.com/pysnmp/docs/hlapi/asyncore/sync/manager/cmdgen/getcmd.html，可以自定义变量名。

Yields:

- **errorIndication** (*str*) – True value indicates SNMP engine error.
- **errorStatus** (*str*) – True value indicates SNMP PDU error.
- **errorIndex** (*int*) – Non-zero value refers to *varBinds [errorIndex-1]*
- **varBinds** (*tuple*) – A sequence of **`ObjectType`** class instances representing MIB variables returned in SNMP response.

其中 varBinds 为元组，包含 SNMP 查询的返回消息。

步骤 7 获取设备主机名

处理 SNMP response 数据，分离主机名。

```
for i in varBinds:
    print (i)
    print (str(i).split('=')[1].strip())
```

varBinds 元组完整信息为：

```
SNMPv2-MIB::sysName.0 = CE1
```

回复的信息包含 MIB 文件、MIB 对象和 ID，最后才是主机名。

使用字符串 split 和 strip 方法，获取单独主机名。

步骤 8 关闭会话

```
dis_this = cli.recv(999999).decode()  #查看脚本交互过程
print (dis_this)
#关闭会话
ssh.close()
```

查看 SSH 通道交互过程，最后关闭会话。

2.5 思考题

1. 如何将查询主机名封装成函数，方便其他程序调用?
2. 如果现网有相当数量设备，如何提高 SNMP 查询效率?

3 NETCONF 配置实验

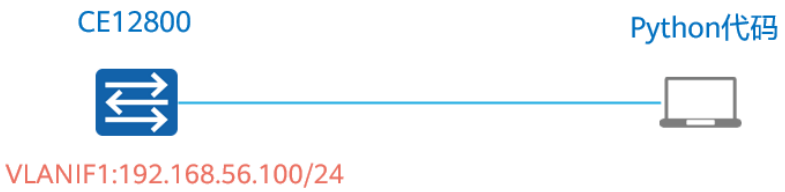
3.1 实验背景

某公司现有一台 CE12800 设备，管理 IP 地址为 192.168.56.100/24，可以 SSH 远程登陆。为了解决通过 SSH 下发配置效率不高的问题，现在要求通过 NETCONF 协议对设备下发配置。

3.1.1 实验目标

- 掌握 paramiko 基本用法
- 掌握 ncclient 基本方法

3.1.2 实验环境准备



本实验需要准备一台网络设备和 Python 编译环境三层互通。

1.配置 CE12800。

```
<HUAWEI>system-view immediately
Enter system view, return user view with return command.
[HUAWEI]sysname CE1
[CE1]interface Vlanif 1
[CE1-Vlanif1]ip add 192.168.56.100 24
[CE1-Vlanif1]quit
```

2.验证可达性。

登录本地 CMD 窗口测试本机到 CE1 的连通性。

```
C:\Users\XXX>ping 192.168.56.100
正在 Ping 192.168.56.100 具有 32 字节的数据:
来自 192.168.56.100 的回复: 字节=32 时间=3ms TTL=255
来自 192.168.56.100 的回复: 字节=32 时间=3ms TTL=255
来自 192.168.56.100 的回复: 字节=32 时间=2ms TTL=255
```

```
来自 192.168.56.100 的回复: 字节=32 时间=4ms TTL=255
192.168.56.100 的 Ping 统计信息:
    数据包: 已发送 = 4，已接收 = 4，丢失 = 0 (0% 丢失)，
往返行程的估计时间(以毫秒为单位):
    最短 = 2ms，最长 = 4ms，平均 = 3ms
```

连通成功。

3.设备 CE1 完成 SSH 配置，用户名 python，密码 Huawei12#$。（参考 1.2.2）

4.准备 NETCONF 预配置脚本，netconf.txt 文件。

新创建 netconf 用户，用户名 netconf，密码 Huawei12#$。完整脚本为：

```
system-view immediately
aaa
local-user netconf password irreversible-cipher Huawei12#$
local-user netconf service-type ssh
local-user netconf level 3
quit
ssh user netconf authentication-type password
ssh user netconf service-type snetconf
snetconf server enable
netconf
protocol inbound ssh port 830
quit
```

3.2 配置思路

1. 完成实验环境准备。
2. 调用 paramiko 登陆设备。
3. 调用 open 打开本地文件 netconf.txt，结合 paramiko 完成设备 NETCONF 配置。
4. 调用 ncclient 通过 NETCONF 配置设备 G1/0/2 接口。其 IP 地址为 192.168.2.1/24，描述信息为 Config by NETCONF。

3.3 完整代码和运行结果

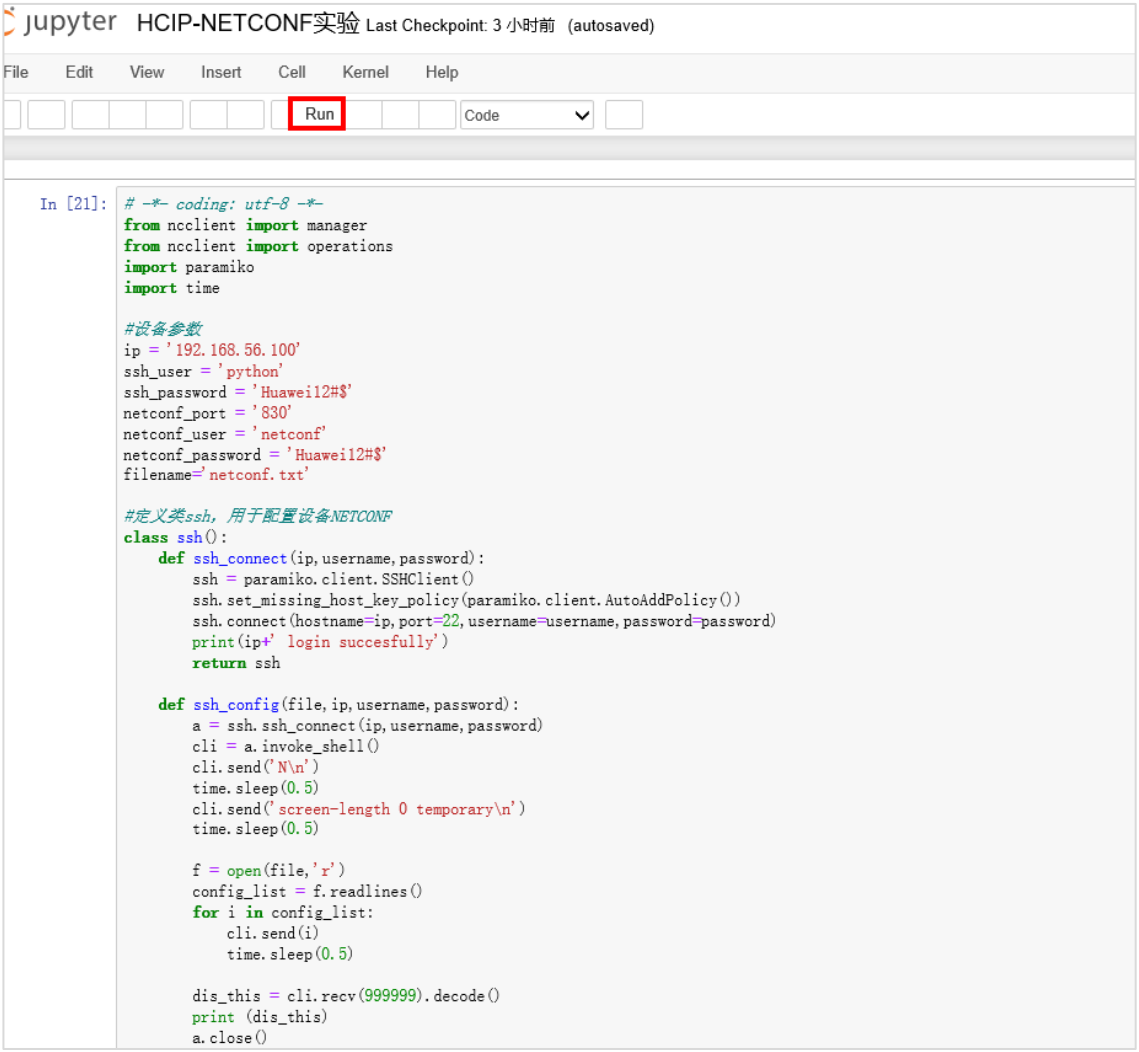
步骤 1 完整代码：

```
# -*- coding: utf-8 -*-
from ncclient import manager
from ncclient import operations
import paramiko
import time

#设备参数
```

```
ip = '192.168.56.100'
ssh_user = 'python'
ssh_password = 'Huawei12#$'
netconf_port = '830'
netconf_user = 'netconf'
netconf_password = 'Huawei12#$'
filename='netconf.txt'

#定义类 ssh，用于配置设备 NETCONF
class ssh():
    def ssh_connect(ip,username,password):
        ssh = paramiko.client.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.client.AutoAddPolicy())
        ssh.connect(hostname=ip,port=22,username=username,password=password)
        print(ip+' login succesfully')
        return ssh

    def ssh_config(file,ip,username,password):
        a = ssh.ssh_connect(ip,username,password)
        cli = a.invoke_shell()
        cli.send('N\n')
        time.sleep(0.5)
        cli.send('screen-length 0 temporary\n')
        time.sleep(0.5)

        f = open(file,'r')
        config_list = f.readlines()
        for i in config_list:
            cli.send(i)
            time.sleep(0.5)

        dis_this = cli.recv(999999).decode()
        print (dis_this)
        a.close()

#定义函数 huawei_connect，用于建立 NETCONF 连接
def huawei_connect(host, port, user, password):
    return manager.connect(host=host,
                           port=port,
                           username=user,
                           password=password,
                           hostkey_verify = False,
                           device_params={'name': "huawei"},
                           allow_agent = False,
                           look_for_keys = False)

#NETCONF 发送 XML 数据，配置设备接口 IP 地址
```

```
CREATE_INTERFACE = '''<config>
        <ethernet xmlns="http://www.huawei.com/netconf/vrp" content-version="1.0" format-
version="1.0">
          <ethernetIfs>
            <ethernetIf operation="merge">
              <ifName>GE1/0/2</ifName>
              <l2Enable>disable</l2Enable>
            </ethernetIf>
          </ethernetIfs>
        </ethernet>
        <ifm xmlns="http://www.huawei.com/netconf/vrp" content-version="1.0" format-version="1.0">
          <interfaces>
            <interface operation="merge">
              <ifName>GE1/0/2</ifName>
              <ifDescr>Config by NETCONF</ifDescr>
              <ifmAm4>
                <am4CfgAddrs>
                  <am4CfgAddr operation="create">
                    <subnetMask>255.255.255.0</subnetMask>
                    <addrType>main</addrType>
                    <ifIpAddr>192.168.2.1</ifIpAddr>
                  </am4CfgAddr>
                </am4CfgAddrs>
              </ifmAm4>
            </interface>
          </interfaces>
        </ifm>
      </config>'''

#主函数，顺序执行
if __name__ == '__main__':
      ssh.ssh_config(filename,ip,ssh_user,ssh_password)
      m = huawei_connect(ip,netconf_port,netconf_user,netconf_password)
      m.edit_config(target='running',config=CREATE_INTERFACE)
```

步骤 2 编译器执行：



步骤 3 输出结果：

编译结果：

```
192.168.56.100 login succesfully


Warning: The initial password poses security risks.
The password needs to be changed. Change now? [Y/N]:N


Info: The max number of VTY users is 5, the number of current VTY users online is 1, and total number of terminal users online is 2.
        The current login time is 2019-11-06 13:08:58.
        The last login time is 2019-11-06 11:48:16 from 192.168.56.1 through SSH.
<CE1>screen-length 0 temporary
Info: The configuration takes effect on the current user terminal interface only.
<CE1>system-view immediately
Enter system view, return user view with return command.
[CE1]aaa
[CE1-aaa]local-user netconf password irreversible-cipher Huawei12#$
Info: A new user is added.
```

```
[CE1-aaa]local-user netconf service-type ssh
[CE1-aaa]local-user netconf level 3
[CE1-aaa]quit
[CE1]ssh user netconf authentication-type password
[CE1]ssh user netconf service-type snetconf
[CE1]snetconf server enable
Info: The SNETCONF server is already started.
[CE1]netconf
[CE1-netconf]protocol inbound ssh port 830
Info: The ssh port 830 service is already started.
[CE1-netconf]quit
[CE1]
```

登陆设备 CE1 查看配置结果：

```
[CE1]interface GE 1/0/2
[CE1-GE1/0/2]display this
#
interface GE1/0/2
 undo portswitch
 description Config by NETCONF
 shutdown
 ip address 192.168.2.1 255.255.255.0
#
return
[CE1-GE1/0/2]
```

NETCONF 配置下发成功。

3.4 代码解析

步骤 1 导入模块

```
# -*- coding: utf-8 -*-
from ncclient import manager
from ncclient import operations
import paramiko
import time
```

导入本脚本使用的所有模块。如果没有安装 ncclient，你可以使用 pip install ncclient 安装。

ncclient 是 python 的 NETCONF 功能模块。本章节简单介绍 ncclient manager 和 operation 的方法以建立和交互 NETCONF，更多 ncclient 信息请参考 https://ncclient.readthedocs.io/en/latest/。

步骤 2 定义设备参数变量

```
#设备参数
ip = '192.168.56.100'
ssh_user = 'python'
```

```
ssh_password = 'Huawei12#$'
netconf_port = '830'
netconf_user = 'netconf'
netconf_password = 'Huawei12#$'
filename='netconf.txt'
```

定义变量对应设备的参数，分别为主机 IP、SSH 用户、SSH 密码、NETCONF 端口、NETCONF 用户名、NETCONF 密码和本地文件名。

步骤 3 完成设备 NETCONF 配置

使用 paramiko 和 open 将本地 netconf.txt 配置脚本命令发送到 CE1 上。本案例使用类封装，方便主函数调用。

```
#定义类 ssh，用于配置设备 NETCONF
class ssh():
```

申明类名为 ssh。类中包含两个方法（函数），ssh_connect()和 ssh_config()，分别用于建立 SSH 连接和发送配置。

```
    def ssh_connect(ip,username,password):
        ssh = paramiko.client.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.client.AutoAddPolicy())
        ssh.connect(hostname=ip,port=22,username=username,password=password)
        print(ip+' login succesfully')
        return ssh
```

ssh 类中定义第一个方法 ssh_connect(ip,username,password)，输入三个参数为 SSH 的 IP、用户名和密码。此函数封装 paramiko 的方法，详细解释参考前面章节。

```
    def ssh_config(file,ip,username,password):
        a = ssh.ssh_connect(ip,username,password)#调用 ssh_connect 赋值给 a
        cli = a.invoke_shell()
        cli.send('N\n')
        time.sleep(0.5)
        cli.send('screen-length 0 temporary\n')
        time.sleep(0.5)

        f = open(file,'r')
        config_list = f.readlines()
        for i in config_list:
            cli.send(i)
            time.sleep(0.5)

        dis_this = cli.recv(999999).decode()
        print (dis_this)
        a.close()
```

ssh 类中第二个方法 ssh_config(file,ip,username,password)。定义四个参数分别为，配置文件路径、SSH 的 IP、用户名和密码。

其中 ssh_config()通过调用 ssh_connect()连接到设备后再发送配置命令。使用 open 函数打开本地 netconf.txt 文件，逐行写入 SSH 通道。最后查看和设备的交互过程，关闭会话。

步骤 4 建立 NETCONF 连接

```
#定义函数 huawei_connect，用于建立 NETCONF 连接
def huawei_connect(host, port, user, password):
    return manager.connect(host=host,
                           port=port,
                           username=user,
                           password=password,
                           hostkey_verify = False,
                           device_params={'name': "huawei"},
                           allow_agent = False,
                           look_for_keys = False)
```

定义函数 huawei_connect(host, port, user, password)。函数输入四个参数为 NETCONF 主机的 IP、端口、NETCONF 用户名和密码。函数返回 ncclient 的 manager.connect 的方法。

Manager.connect 作用是建立 NETCONF 连接。参数由 RFC4741 定义，其中 device_params 对于华为设备有两个选择。可以配置为 huawei 或者 huaweiyang，分别对应 IETF YANG 和华为的 YANG 模型。

步骤 5 构建 XML 配置文件

参考华为官网《NETCONF Schema API 参考》，构造 XML 配置文件。
https://support.huawei.com/enterprise/zh/switches/cloudengine-12800-pid-7542409

```
#NETCONF 发送 XML 数据，配置设备接口 IP 地址
CREATE_INTERFACE = '''<config>
      <ethernet xmlns="http://www.huawei.com/netconf/vrp" content-version="1.0" format-version="1.0">
        <ethernetIfs>
          <ethernetIf operation="merge">
            <ifName>GE1/0/2</ifName>
            <l2Enable>disable</l2Enable>
          </ethernetIf>
        </ethernetIfs>
      </ethernet>
      <ifm xmlns="http://www.huawei.com/netconf/vrp" content-version="1.0" format-version="1.0">
        <interfaces>
          <interface operation="merge">
            <ifName>GE1/0/2</ifName>
            <ifDescr>Config by NETCONF</ifDescr>
            <ifmAm4>
              <am4CfgAddrs>
                <am4CfgAddr operation="create">
                  <subnetMask>255.255.255.0</subnetMask>
                  <addrType>main</addrType>
                  <ifIpAddr>192.168.2.1</ifIpAddr>
```

```
                    </am4CfgAddr>
                  </am4CfgAddrs>
                </ifmAm4>
              </interface>
            </interfaces>
          </ifm>
        </config>'''
```

NETCONF 通过 XML 文件传递配置信息。XML 是一种非常常用的文本格式，可以<>不断嵌套展开数据。完整的 NETCONF 会话有传输层、消息层、操作层和内容层。在当前 XML 配置文件中传递的仅包含操作层和内容层。

NETCONF 基本操作包括 get-config、get、edit-config、copy-config、delete-config、lock、unlock、close-session 和 kill session。例如本例的操作层信息为 edit-config，对应具体 operation 属性为 merge，在数据库中修改存在或不存在的目标数据，如果目标数据不存在则创建，如果目标数据存在则修改。

NETCONF 内容层则为对具体参数的修改。本例中对应首先关闭接口 GE 1/0/2 的 L2 功能，对应命令 undo portswitch，然后修改 description 信息为 Config by NETCONF，最后配置接口 IP 地址为 192.168.2.1/24。

步骤 6 运行主函数

```
#主函数，顺序执行
if __name__ == '__main__':
    ssh.ssh_config(filename,ip,ssh_user,ssh_password)
    m = huawei_connect(ip,netconf_port,netconf_user,netconf_password)
    m.edit_config(target='running',config=CREATE_INTERFACE)
```

运行主函数。

首先执行 ssh.ssh_config(filename,ip,ssh_user,ssh_password)，调用 ssh 类下 ssh_config 方法。输入参数为步骤 2 中定义的变量。

然后执行 huawei_connect(ip,netconf_port,netconf_user,netconf_password)赋值给 m。输入 NETCONF 参数，建立 NETCONF 连接。

最后执行 m.edit_config(target='running',config=CREATE_INTERFACE)。将步骤 5 构造的 XML 配置文件通过 edit_config 方法发送到设备 running 配置文件。现在你可以登陆设备验证结果。配置成功下发。

3.5 思考题

1. 为什么要使用 if 构建主函数?
2. NETCONF 可以 get 操作可以查询设备信息和 SNMP 有什么区别?

4 配置文件对比实验

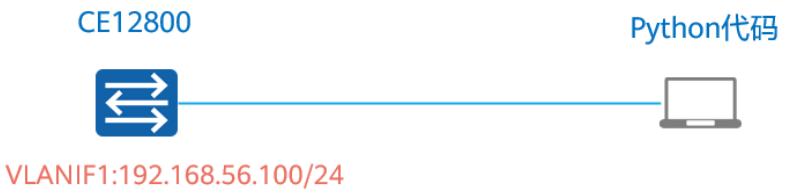
4.1 实验背景

某公司现在有多台网络设备（本例 1 台）。为了更好的实现配置监控，现在需要使用代码实现每台设备的配置对比，并输出配置对比结果。

4.1.1 实验目标

- 掌握 difflib 实现文本对比方法
- 掌握正则表达式处理数据
- 掌握 paramiko 配置设备

4.1.2 实验环境准备



本实验需要准备一台网络设备和 Python 编译环境三层互通。

1.配置 CE12800。

```
<HUAWEI>system-view immediately
Enter system view, return user view with return command.
[HUAWEI]sysname CE1
[CE1]interface Vlanif 1
[CE1-Vlanif1]ip add 192.168.56.100 24
[CE1-Vlanif1]quit
```

2.验证可达性。

登录本地 CMD 窗口测试本机到 CE1 的连通性。

```
C:\Users\XXX>ping 192.168.56.100
正在 Ping 192.168.56.100 具有 32 字节的数据:
来自 192.168.56.100 的回复: 字节=32 时间=3ms TTL=255
来自 192.168.56.100 的回复: 字节=32 时间=3ms TTL=255
```

```
来自 192.168.56.100 的回复: 字节=32 时间=2ms TTL=255
来自 192.168.56.100 的回复: 字节=32 时间=4ms TTL=255
192.168.56.100 的 Ping 统计信息:
    数据包: 已发送 = 4，已接收 = 4，丢失 = 0 (0% 丢失)，
往返行程的估计时间(以毫秒为单位):
    最短 = 2ms，最长 = 4ms，平均 = 3ms
```

连通成功。

3.设备 CE1 完成 SSH 配置，用户名 python，密码 Huawei12#$。（参考 1.2.2）

4.2 完整代码和运行结果

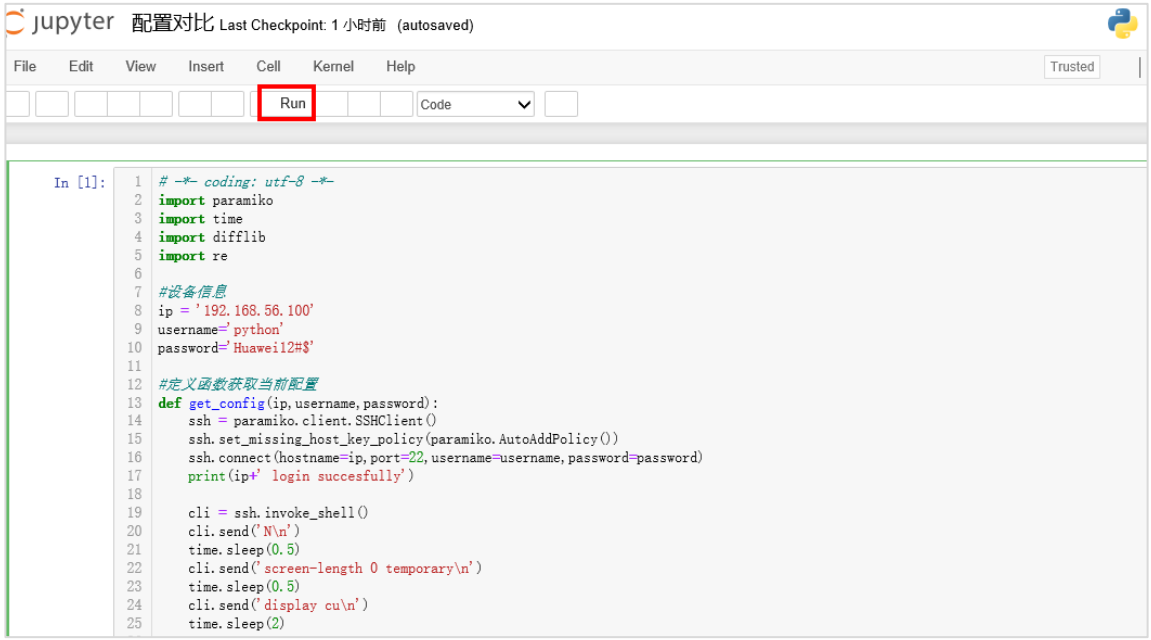
步骤 1 完整代码：

```
# -*- coding: utf-8 -*-
import paramiko
import time
import difflib
import re

#设备信息
ip = '192.168.56.100'
username='python'
password='Huawei12#$'

#定义函数获取当前配置
def get_config(ip,username,password):
    ssh = paramiko.client.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh.connect(hostname=ip,port=22,username=username,password=password)
    print(ip+' login succesfully')

    cli = ssh.invoke_shell()
    cli.send('N\n')
    time.sleep(0.5)
    cli.send('screen-length 0 temporary\n')
    time.sleep(0.5)
    cli.send('display cu\n')
    time.sleep(2)

    dis_cu = cli.recv(999999).decode()
    return (dis_cu)
    ssh.close()

#定义函数 ssh_config，将脚本写入设备
```

```
def ssh_config(file,ip,username,password):
    ssh = paramiko.client.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh.connect(hostname=ip,port=22,username=username,password=password)
    print(ip+' login succesfully')

    cli = ssh.invoke_shell()
    cli.send('N\n')
    time.sleep(0.5)
    cli.send('screen-length 0 temporary\n')
    time.sleep(0.5)

    f = open(file,'r')
    config_list = f.readlines()
    for i in config_list:
        cli.send(i)
        time.sleep(0.5)

    dis_this = cli.recv(999999).decode()
    #print (dis_this)
    ssh.close()

#调用 get_config 赋值给 output
output = get_config(ip,username,password)
#print (output)

#数据处理，使用正则表达式仅获取配置信息
config = re.findall(r'(<CE1>display cu[\d\D]+<CE1>$)',output)
#print (config)

#保存配置到本地文件 file1
with open(r'D:\Config\file1','w') as f:
    f.writelines(config[0])

#调用 ssh_config，将 netconf.txt 配置写入设备
ssh_config('netconf.txt',ip,username,password)

#再次读取配置，保存到本地为 file2
output = get_config(ip,username,password)
config = re.findall(r'(<CE1>display cu[\d\D]+<CE1>$)',output)

with open(r'D:\Config\file2','w') as f:
    f.writelines(config[0])

#配置对比
d = difflib.HtmlDiff()
```

```
#定义函数读取文件
def read_file(filename):
    try:
        with open(filename,'r') as f:
            return f.readlines()
    except IOError:
        print('%s 未找到该文件' % filename)
        sys.exit(1)

#定义函数 compare_files，做配置对比，并保存文件为 result.html
def compare_files(file1,file2,out_file):
    file1_content = read_file(file1)
    file2_content = read_file(file2)
    d = difflib.HtmlDiff()
    result = d.make_file(file1_content,file2_content)
    with open(r'D:\Config\result.html','w') as f:
        f.writelines(result)
    print ()

#调用 compare_files
compare_files(r'D:\Config\file1',r'D:\Config\file2',r'D:\Config\result.html')
```

步骤 2 编译执行：

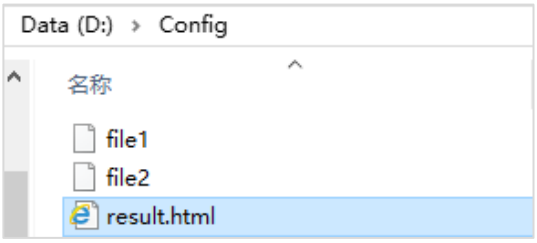


步骤 3 输出结果：

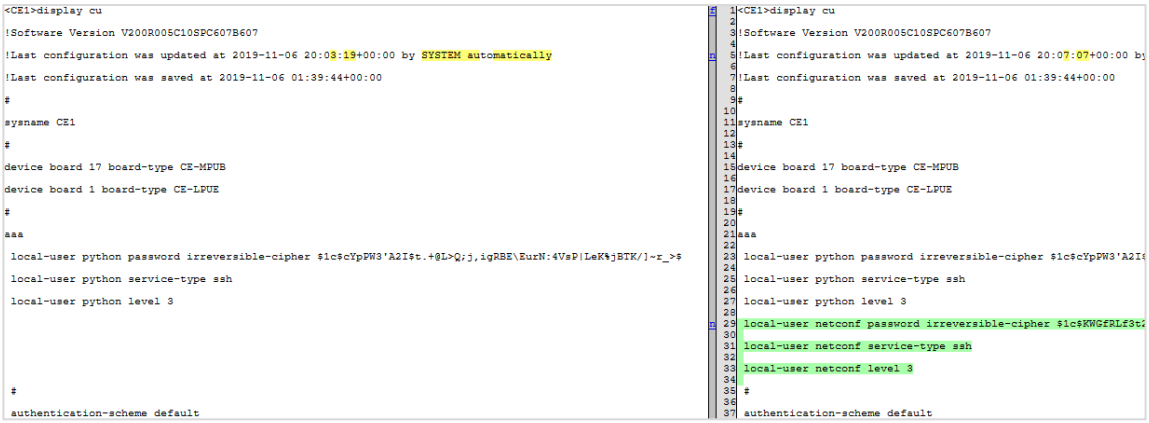
编译结果：

```
192.168.56.100 login successfully
192.168.56.100 login successfully
192.168.56.100 login successfully
```

目录下的配置文件和对比文件：



打开 html 文件：



4.3 代码解析

步骤 1 导入模块

```
# -*- coding: utf-8 -*-
import paramiko
import time
import difflib
import re
```

导入本脚本使用的所有模块。如果没有模块请使用 pip install [模块名] 安装。

本案例中 difflib 模块用于实现文本对比。re 模块用于使用正则表达式处理数据。

步骤 2 定义设备参数变量

```
#设备信息
ip = '192.168.56.100'
username='python'
password='Huawei12#$'
```

定义变量对应设备的参数，分别为主机 IP、SSH 用户、SSH 密码。

步骤 3 定义函数 get_config()

定义函数 get_config()，用于获取设备当前配置。

```
#定义函数获取当前配置
```

```
def get_config(ip,username,password):
    ssh = paramiko.client.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh.connect(hostname=ip,port=22,username=username,password=password)
    print(ip+' login succesfully')

    cli = ssh.invoke_shell()
    cli.send('N\n')
    time.sleep(0.5)
    cli.send('screen-length 0 temporary\n')
    time.sleep(0.5)
    cli.send('display cu\n')
    time.sleep(2)

    dis_cu = cli.recv(999999).decode()
    return (dis_cu)
    ssh.close()
```

get_config(ip,username,password)函数，输入参数 IP，用户名和密码。函数内调用 paramiko 登陆设备，执行 display current-configuration 收集设备回显信息。最后将回显信息返回。具体 paramiko 方法解析查看 Paramiko 登陆设备实验。

步骤 4 定义 ssh_config()函数

定义 ssh_config()函数，用于将配置脚本发送到设备。

```
#定义函数 ssh_config，将脚本写入设备
def ssh_config(file,ip,username,password):
    ssh = paramiko.client.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh.connect(hostname=ip,port=22,username=username,password=password)
    print(ip+' login succesfully')

    cli = ssh.invoke_shell()
    cli.send('N\n')
    time.sleep(0.5)
    cli.send('screen-length 0 temporary\n')
    time.sleep(0.5)
#读取 file 文件，将文件逐行写入设备
    f = open(file,'r')
    config_list = f.readlines()
    for i in config_list:
        cli.send(i)
        time.sleep(0.5)

    dis_this = cli.recv(999999).decode()
#     print (dis_this)
    ssh.close()
```

4.4 思考题

1. 为什么编译结果输出了三次 192.168.56.100 login successfully，分别是为什么？
2. 本案例脚本有部分冗余，哪些地方可以优化？
3. 是否可以定时的保存配置文件和进行配置对比？是否可以用其他方法更简单的获取配置文件？

5 gRPC 远程查询配置实验

5.1 实验背景

某公司现有一台 CE12800 设备，管理 IP 地址为 192.168.56.100。因为特殊需求，需要实现客户端通过 gRPC 调用服务器查询并返回设备的当前配置。

5.1.1 实验目标

- 掌握自定义 proto 文件
- 掌握 gRPC 服务端代码编写
- 掌握 gRPC 客户端代码编写

5.1.2 实验环境准备

使用模拟器或者真机环境准备实验环境。客户端和服务端代码均在本机编写。

基本环境搭建同《 SSH 实验 》。

1.设备 CE1 完成 SSH 配置，用户名 python，密码 Huawei12#$。

2.配置设备管理地址 192.168.56.100/24。

3.Juypter Notebook 或其他编译器上创建两个 Python 文件，分别做客户端和服务端。

5.2 配置思路

1. 编写.proto 文件。定义 gRPC 请求和响应的服务和方法。

2. 根据.proto 文件生成客户端和服务端的 Python 代码。
3. 编写服务器代码。
4. 编写客户端代码。

5.3 配置过程和完整代码

5.3.1 编写.proto 文件

```
syntax = "proto3";
package get_config;
// The get_config service definition.
service get_config {
  // RPC 请求和响应
  rpc Login_info (Request) returns (Reply) {}
}
// The request message containing the login information.
message Request {
  string host = 1;
  string username = 2;
  string password = 3;
}
// The response message containing the string reply.
message Reply {
  string message = 1;
}
```

保存为 get_config.proto 格式。

在.proto 文件中定义了 get_config 服务。请求方法 Login_info()，输入参数 Request 包含三个字符串类型参数 host、username 和 password。返回参数 Reply 包含一个字符串参数 message。

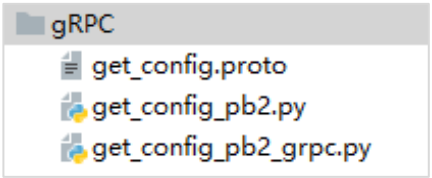
5.3.2 生成客户端和服务端代码

参考 https://grpc.io/docs/tutorials/basic/python/，安装 grpcio-tools 和运行生成代码命令。

CMD 在.proto 文件所在目录执行如下命令：

```
C:\Users\Richard\HCIP\gRPC>python -m grpc_tools.protoc -I./ --python_out=. --
grpc_python_out=. ./get_config.proto
```

执行后将自动生成客户端和服务端代码：



5.3.3 服务器端完整代码

```
from concurrent import futures
import time
import grpc
import get_config_pb2
import get_config_pb2_grpc
import paramiko

class Display_Config(get_config_pb2_grpc.get_configServicer):
    #调用 paramiko 登陆设备获取当前配置
    def Login_info(self, request, context):
        ssh = paramiko.client.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(hostname=request.host, port=22, username=request.username,
password=request.password)
        cli = ssh.invoke_shell()
        cli.send('N\n')
        time.sleep(0.5)
        cli.send('screen-length 0 temporary\n')
        time.sleep(0.5)
        cli.send('display cu\n')
        time.sleep(3)
        data = cli.recv(999999).decode()
        ssh.close()
        #返回回显的配置信息
        return get_config_pb2.Reply(message=data)

def serve():
    #创建 gRPC 服务
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    #从定义的服务中部署 gRPC servicer
    get_config_pb2_grpc.add_get_configServicer_to_server(Display_Config(),server)
    #启动服务器
    server.add_insecure_port('localhost:8080')
    server.start()
    _ONE_DAY_IN_SECONDS = 60 * 60 * 24
    try:
        while True:
            time.sleep(_ONE_DAY_IN_SECONDS)
    except KeyboardInterrupt:
        server.stop()

if __name__ == "__main__":
    serve()
```

运行服务器，将持续监听。

5.3.4 客户端完整代码

```
import grpc
import get_config_pb2
import get_config_pb2_grpc

def run():
    #客户端实例化 stub
    connect = grpc.insecure_channel('localhost:8080')
    stub = get_config_pb2_grpc.get_configStub(channel=connect)
    #通过 stub 调用服务端的 Login_info 方法
    response =
stub.Login_info(get_config_pb2.Request(host='192.168.56.100',username='python',password='Huawei12#
$'))
    print (response.message)

if __name__ == "__main__":
    run()
```

运行客户端，输入需要登陆的设备 IP 地址、用户和端口。服务器运行返回信息：

```
Warning: The initial password poses security risks.
The password needs to be changed. Change now? [Y/N]:N

Info: The max number of VTY users is 5, the number of current VTY users online is 1, and total number
of terminal users online is 2.
        The current login time is 2020-01-31 15:58:01.
        The last login time is 2020-01-31 15:57:20 from 192.168.56.1 through SSH.
<CE1>screen-length 0 temporary
Info: The configuration takes effect on the current user terminal interface only.
<CE1>display cu
!Software Version V200R005C10SPC607B607
!Last configuration was updated at 2020-01-31 15:31:54+00:00 by SYSTEM automatically
!Last configuration was saved at 2019-11-06 01:39:44+00:00
#
sysname CE1
#
device board 17 board-type CE-MPUB
device board 1 board-type CE-LPUE
#
aaa
 local-user python password irreversible-cipher
$1c$cYpPW3'A2I$t.+@L>Q;j,igRBE\EurN:4VsP|LeK%jBTK/]~r_>$
 local-user python service-type ssh
 local-user python level 3
 #
 authentication-scheme default
 #
 authorization-scheme default
```

```
 #
 accounting-scheme default
 #
 domain default
 #
 domain default_admin
#
interface Vlanif1
 ip address 192.168.56.100 255.255.255.0
#
interface MEth0/0/0
 undo shutdown
#
interface GE1/0/0
 undo shutdown
#
interface GE1/0/1
 shutdown
#
interface GE1/0/2
 shutdown
#
interface GE1/0/3
 shutdown
#
interface GE1/0/4
 shutdown
#
interface GE1/0/5
 shutdown
#
interface GE1/0/6
 shutdown
#
interface GE1/0/7
 shutdown
#
interface GE1/0/8
 shutdown
#
interface GE1/0/9
 shutdown
#
interface NULL0
#
snmp-agent
snmp-agent local-engineid 800007DB03707BE82F1330
#
```

```
snmp-agent sys-info version v3
snmp-agent group v3 dc-admin privacy read-view rd write-view wt notify-view nt
#
snmp-agent mib-view included nt iso
snmp-agent mib-view included rd iso
snmp-agent mib-view included wt iso
snmp-agent mib-view included iso-view iso
snmp-agent usm-user v3 admin
snmp-agent usm-user v3 admin group dc-admin
snmp-agent usm-user v3 admin authentication-mode sha
cipher %^%#FQ]J>Ba"e*F{#/+rx$UUZ2^:BaG>V/LoKc8No%DE%^%#
snmp-agent usm-user v3 admin privacy-mode aes128
cipher %^%#BEj8L``EE,88ziP^'jFW5A+EB"uqg'wdaS+Y9$F.%^%#
#
snmp-agent trap source Vlanif1
#
stelnet server enable
ssh authorization-type default aaa
#
ssh server cipher aes256_gcm aes128_gcm aes256_ctr aes192_ctr aes128_ctr aes256_cbc aes128_cbc
3des_cbc
#
ssh server dh-exchange min-len 1024
#
ssh client cipher aes256_gcm aes128_gcm aes256_ctr aes192_ctr aes128_ctr aes256_cbc aes128_cbc
3des_cbc
#
user-interface con 0
#
user-interface vty 0 4
 authentication-mode aaa
 user privilege level 3
#
vm-manager
#
return
<CE1>

Process finished with exit code 0
```

5.4 代码解析

5.4.1 服务器代码

编写 gRPC 服务器代码有三个步骤：

第一：调用.proto 定义的服务。生成的.py 文件已经具体实现了服务，服务器只需要调用即可。

第二：编写服务器需要实现的功能代码。

第三：运行服务器。

创建子类 Display_Config()，继承 get_config_pb2_grpc 类的属性和方法。

```
class Display_Config(get_config_pb2_grpc.get_configServicer):
```

在函数 Login_info 定义了为 RPC 的请求和响应。

```
def Login_info(self, request, context):
```

Request 的结构在 protocol buffers 中已定义，有 host、username 和 password 三个属性。此时接收客户端传递过来的参数，调用 paramiko 模块，执行 SSH 登陆设备和查询当前配置的指令。最后将数据返回。

```
    #调用 paramiko 登陆设备获取当前配置
        ssh = paramiko.client.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(hostname=request.host, port=22, username=request.username,
password=request.password)
        cli = ssh.invoke_shell()
        cli.send('N\n')
        time.sleep(0.5)
        cli.send('screen-length 0 temporary\n')
        time.sleep(0.5)
        cli.send('display cu\n')
        time.sleep(3)
        data = cli.recv(999999).decode()
        ssh.close()
        #返回回显的配置信息
        return get_config_pb2.Reply(message=data)
```

创建 serve()函数实例化 gRPC 服务，被主函数调用执行。

此时启动服务器 8080 端口，在一天的时间内不断的侦听此端口。等待客户端调用此服务。

```
def serve():
    #创建 gRPC 服务
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    #从定义的服务中部署 gRPC servicer
    get_config_pb2_grpc.add_get_configServicer_to_server(Display_Config(),server)
    #启动服务器
    server.add_insecure_port('localhost:8080')
    server.start()
    _ONE_DAY_IN_SECONDS = 60 * 60 * 24
    try:
        while True:
            time.sleep(_ONE_DAY_IN_SECONDS)
    except KeyboardInterrupt:
```

```
            server.stop()

if __name__ == "__main__":
    serve()
```

5.4.2 客户端代码

编写 gRPC 客户端代码有两个步骤：

第一：创建存根（stub），以后面调用服务。

第二：通过存根调用服务方法。

创建存根(stub)，用于连接服务端。

```
def run():
    #客户端实例化 stub
    connect = grpc.insecure_channel('localhost:8080')
    stub = get_config_pb2_grpc.get_configStub(channel=connect)
```

通过 stub 调用服务端的 Login_info 方法。

本例中输入需要查询的设备 IP 地址、用户名和密码，即可返回设备当前配置。

```
    response =
stub.Login_info(get_config_pb2.Request(host='192.168.56.100',username='python',password='Huawei12#
$'))
    print (response.message)

if __name__ == "__main__":
    run()
```

实现了本实验中的查询配置在服务器上运行，将结果返回给客户端。

5.5 思考题

gRPC 的好处是什么？你还能想到有哪些使用场景？

6 Telemetry 配置实验

6.1 实验说明

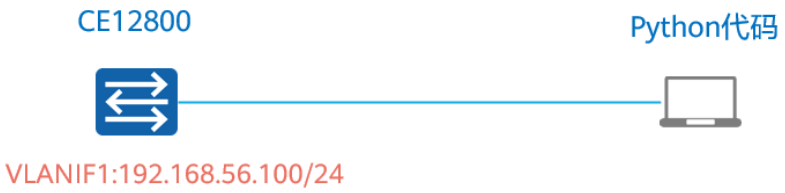
某公司现有一台 CE12800 设备，管理 IP 地址为 192.168.56.100。为了更好的采集设备性能数据，现在要求通过 Telemetry 静态订阅方式，设备推送 CPU 信息到服务端。

6.1.1 实验目标

- 掌握华为 Telemetry 静态采集方式服务端代码编写
- 掌握华为 Proto 文件中定义的常见方法

6.1.2 实验环境准备

使用模拟器或者真机准备实验环境。本地 PC 运行 Python 脚本为服务端（本例地址为 192.168.56.1），交换机推送数据到服务端（本例 CE1 地址为 192.168.56.100）。



1.配置 CE12800。

```
<HUAWEI>system-view immediately
Enter system view, return user view with return command.
[HUAWEI]sysname CE1
[CE1]interface Vlanif 1
[CE1-Vlanif1]ip add 192.168.56.100 24
[CE1-Vlanif1]quit
```

2.验证可达性。

登录本地 CMD 窗口测试本机到 CE1 的连通性。

```
C:\Users\XXX>ping 192.168.56.100
正在 Ping 192.168.56.100 具有 32 字节的数据:
来自 192.168.56.100 的回复: 字节=32 时间=3ms TTL=255
来自 192.168.56.100 的回复: 字节=32 时间=3ms TTL=255
```

```
来自 192.168.56.100 的回复: 字节=32 时间=2ms TTL=255
来自 192.168.56.100 的回复: 字节=32 时间=4ms TTL=255
192.168.56.100 的 Ping 统计信息:
    数据包: 已发送 = 4，已接收 = 4，丢失 = 0 (0% 丢失)，
往返行程的估计时间(以毫秒为单位):
    最短 = 2ms，最长 = 4ms，平均 = 3ms
```

连通成功。

3.设备 CE1 完成 SSH 配置，用户名 python，密码 Huawei12#$。（参考 1.2.2）

6.2 配置思路

1. 搭建环境，完成实验环境准备。
2. 配置设备 Telemetry 静态订阅方式相关配置，包括采集内容、推送对象和推送间隔。
3. 官网下载华为设备对应发布的.proto 文件。（获取 Proto 文件的方式：Proto 文件的下载方式同系统软件类似。访问华为企业用户技术支（http://support.huawei.com/enterprise）或运营商用户技术支持网站（http://support.huawei.com/carrier），搜索相应的设备型号及版本。然后进入软件下载页面获取相应版本的 Proto 文件。）编译 proto 文件得到服务端调用方法。
4. 编写服务端代码，监听指定端口获取数据。
5. 根据上送数据的不同，选择对应的方法对数据内容进行解码。

6.3 配置过程和完整代码

6.3.1 交换机配置

保证交换机与服务端三层可达后，在交换机上完成 Telemetry 静态订阅方式配置。

步骤 1 进入 Telemetry 视图

```
<CE1> system-view immediately
Enter system view, return user view with return command.
[CE1] telemetry
[CE1-telemetry]
```

步骤 2 配置设备推送目标

本例中创建目标组 Dest1。推送目标 IP 地址为 192.168.56.1，端口为 20000。

```
[CE1-telemetry] destination-group Dest1
[CE1-telemetry-destination-group-Dest1] ipv4-address 192.168.56.1 port 20000 protocol grpc no-tls
```

步骤 3 配置设备采样数据

当用户配置 Telemetry 静态订阅采样数据时，需要创建采样传感器组，并指定好采样路径。本例中创建采样组 Sensor1。采样路径为 CPU 信息。

```
[CE1-telemetry] sensor-group Sensor1
[CE1-telemetry-sensor-group-Sensor1] sensor-path huawei-devm:devm/cpuInfos/cpuInfo
```

步骤 4 创建静态订阅

创建订阅，将配置好的上送目标组和采样传感器组进行关联，完成数据上送。本例中关联目标组 Dest1 与传感器组 Sensor1，并设置采样间隔为 1000ms。

```
[CE1-telemetry]subscription Sub1
[CE1-telemetry-subscription-Sub1]destination-group Dest1
[CE1-telemetry-subscription-Sub1]sensor-group Sensor1 sample-interval 1000
```

至此，设备将持续向目标推送数据。

6.3.2 编译 proto 文件

接下来需要从 .proto 的服务定义中生成 gRPC 客户端和服务器端的接口。你可以通过 protocol buffer 的编译器 protoc 以及一个特殊的 gRPC Python 插件来完成。确保你已经安装了 protoc 并且按照 gRPC Python 插件。更多信息可以参考 https://grpc.io/docs/tutorials/basic/python/。

本例中，我们使用 run_codegen.py 脚本编译 proto 文件。注意将所有 .proto 文件放入 /protos 目录。本脚本将一次编译 huawei-grpc-dialout.proto、huawei-telemetry.proto 和 huawei-devm.proto。如果你想编译更多，可以按照相同格式增加。

```
"""Generates protocol messages and gRPC stubs."""

from grpc_tools import protoc

protoc.main(
    (
        '',
        '-I./protos',
        '--python_out=.',
        '--grpc_python_out=.',
        './protos/huawei-grpc-dialout.proto', #文件路径
    )
)
protoc.main(
    (
        '',
        '-I./protos',
        '--python_out=.',
        '--grpc_python_out=.',
        './protos/huawei-telemetry.proto',
    )
)
```

```
protoc.main(
    (
        '',
        '-I./protos',
        '--python_out=.',
        '--grpc_python_out=.',
        './protos/huawei-devm.proto',
    )
)
```

编译完成后将在 run_codegen.py 目录下生成如下 Python 文件：

6.3.3 Python 服务端完整代码

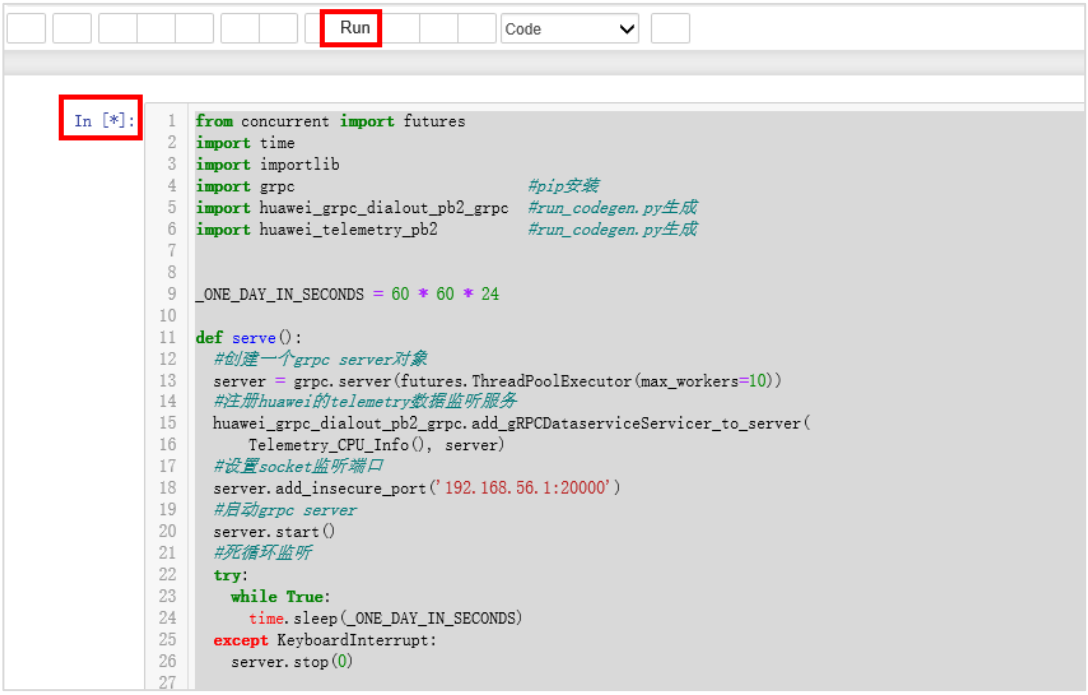
步骤 1 完整代码：

```
from concurrent import futures
import time
import importlib
import grpc                                    #pip 安装
import huawei_grpc_dialout_pb2_grpc    #run_codegen.py 生成
import huawei_telemetry_pb2                 #run_codegen.py 生成


_ONE_DAY_IN_SECONDS = 60 * 60 * 24

def serve():
  #创建一个 grpc server 对象
  server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
  #注册 huawei 的 telemetry 数据监听服务
  huawei_grpc_dialout_pb2_grpc.add_gRPCDataserviceServicer_to_server(
      Telemetry_CPU_Info(), server)
  #设置 socket 监听端口
  server.add_insecure_port('192.168.56.1:20000')
  #启动 grpc server
  server.start()
  #死循环监听
  try:
    while True:
```

```
        time.sleep(_ONE_DAY_IN_SECONDS)
    except KeyboardInterrupt:
        server.stop(0)

#创建类继承 huawei_grpc_dialout_pb2_grpc 中 Servicer 方法
class Telemetry_CPU_Info(huawei_grpc_dialout_pb2_grpc.gRPCDataserviceServicer):
    def __init__(self):
        return

    def dataPublish(self, request_iterator, context):
        for i in request_iterator:
            print ('############ start ############\n')
            telemetry_data = huawei_telemetry_pb2.Telemetry.FromString(i.data)
            print (telemetry_data)

            for row_data in   telemetry_data.data_gpb.row:
                print ('-----------------')
                print ('The proto path is :'+telemetry_data.proto_path)
                print ('-----------------')
                module_name = telemetry_data.proto_path.split('.')[0]
                root_class   = telemetry_data.proto_path.split('.')[1]

#动态加载 telemetry 获取数据的对应模块，本例中为
                decode_module = importlib.import_module( module_name+'_pb2')
                print (decode_module)
#定义解码方法：getattr 获取动态加载的模块中的属性值，调用此属性的解码方法 FromString
                decode_func = getattr(decode_module,root_class).FromString

                print ('----------- content is -----------\n')
#  将 row_data 中的 content 中的内容使用此方法解码，并输出
                print (decode_func(row_data.content))
                print ('----------- done -----------')


if __name__ == '__main__':
    serve()
```

步骤 2 编译器执行：

没有中断情况下，服务端将持续监听。




```
from concurrent import futures
import time
import importlib
import grpc                              #pip安装
import huawei_grpc_dialout_pb2_grpc  #run_codegen.py生成
import huawei_telemetry_pb2          #run_codegen.py生成


_ONE_DAY_IN_SECONDS = 60 * 60 * 24

def serve():
  #创建一个grpc server对象
  server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
  #注册huawei的telemetry数据监听服务
  huawei_grpc_dialout_pb2_grpc.add_gRPCDataserviceServicer_to_server(
      Telemetry_CPU_Info(), server)
  #设置socket监听端口
  server.add_insecure_port('192.168.56.1:20000')
  #启动grpc server
  server.start()
  #死循环监听
  try:
    while True:
      time.sleep(_ONE_DAY_IN_SECONDS)
  except KeyboardInterrupt:
    server.stop(0)
```

步骤 3 输出结果：

数据会根据设置间隔持续推送。本例以一次输出为例。

```
############ start ############

node_id_str: "CE1"
subscription_id_str: "Sub1"
sensor_path: "huawei-devm:devm/cpuInfos/cpuInfo"
collection_id: 260
collection_start_time: 1580278055740
msg_timestamp: 1580278055824
data_gpb {
  row {
    timestamp: 1580278055740
    content: "*\022\n\020\"\0011\010\201\200\204\010(\006\030Z0K\020\010"
  }
  row {
    timestamp: 1580278055740
    content: "*\023\n\021\"\00217\010\201\200\304\010(\006\030Z0K\020\010"
  }
}
collection_end_time: 1580278055740
current_period: 60000
except_desc: "OK"
product_name: "CE12800"
proto_path: "huawei_devm.Devm"

----------
```

```
The proto path is :huawei_devm.Devm
-----------------
<module 'huawei_devm_pb2' from 'D:\\10 Python Learning\\Telemetry\\huawei_devm_pb2.py'>
----------- content is -----------

cpuInfos {
  cpuInfo {
    entIndex: 16842753
    interval: 8
    ovloadThreshold: 90
    position: "1"
    systemCpuUsage: 6
    unovloadThreshold: 75
  }
}

----------- done -----------------
-----------------
The proto path is :huawei_devm.Devm
-----------------
<module 'huawei_devm_pb2' from 'D:\\10 Python Learning\\Telemetry\\huawei_devm_pb2.py'>
----------- content is -----------

cpuInfos {
  cpuInfo {
    entIndex: 17891329
    interval: 8
    ovloadThreshold: 90
    position: "17"
    systemCpuUsage: 6
    unovloadThreshold: 75
  }
}

----------- done -----------------
```

6.4 代码解析

服务端的代码将调用编译 proto 文件生成的 Python 文件。run_codegen.py 脚本简单不做更多介绍。编写服务器代码一部分的重要工作就是学会调用这些生成的类和方法。

步骤 1 导入模块

```
from concurrent import futures
import time
import importlib
import grpc                                    #pip 安装
```

```
import huawei_grpc_dialout_pb2_grpc    #run_codegen.py 生成
import huawei_telemetry_pb2              #run_codegen.py 生成
```

导入本脚本使用的所有模块。如果没有安装 gRPC 请参考如下链接完成准备工作：

https://grpc.io/docs/tutorials/basic/python/

concurrent.futures 作用是实现服务端的多进程/多线程。

importlib 作用是动态导入模块。

步骤 2 主函数

```
if __name__ == '__main__':
    serve()
```

程序首先运行主函数。主函数中调用了 serve()。

步骤 3 serve()函数

serve()函数是服务器主体，进行线程设置、监听 IP 设置和服务启停等操作。

```
#定义变量，一天 60*60*24 秒
_ONE_DAY_IN_SECONDS = 60 * 60 * 24

def serve():
    #创建一个 grpc server 对象，允许最大 10 线程
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    #注册 huawei 的 telemetry 数据监听服务
    huawei_grpc_dialout_pb2_grpc.add_gRPCDataserviceServicer_to_server(
        Telemetry_CPU_Info(), server)
    #设置 socket 监听端口
    server.add_insecure_port('192.168.56.1:20000')
    #启动 grpc server
    server.start()
    #死循环监听
    try:
        while True:
            time.sleep(_ONE_DAY_IN_SECONDS)
    except KeyboardInterrupt:
        server.stop(0)
```

serve()函数中首先使用 futures 允许服务端接收最大为 10 的多线程，并赋值给 server。

```
server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
```

然后调用 huawei_grpc_dialout_pb2_grpc 中的 add_gRPCDataserviceServicer_to_server 方法。输入的 Telemetry_CPU_Info()类在后续定义。(可以打开 huawei_grpc_dialout_pb2_grpc.py 文件查看具体函数)

```
huawei_grpc_dialout_pb2_grpc.add_gRPCDataserviceServicer_to_server(
        Telemetry_CPU_Info(), server)
```

server()设置监听的地址和端口。本例使用 192.168.56.1 地址与交换机通信。

```
server.add_insecure_port('192.168.56.1:20000')
```

启动 gRPC 服务。使用 While 函数在 time.sleep()时间内持续运行。

```
server.start()
  #死循环监听
  try:
    while True:
      time.sleep(_ONE_DAY_IN_SECONDS)
  except KeyboardInterrupt:
    server.stop(0)
```

步骤 4 获取 Telemetry 数据

serve()函数中调用了 Telemetry_CPU_Info()类，它的作用是具体实现 Telemtry 数据的获取和解析。首先来查看如何获取数据。

```
#创建类继承 huawei_grpc_dialout_pb2_grpc 中 Servicer 方法
class Telemetry_CPU_Info(huawei_grpc_dialout_pb2_grpc.gRPCDataserviceServicer):
    def __init__(self):
        return

    def dataPublish(self, request_iterator, context):
        for i in request_iterator:
            print ('############ start ############\n')
            telemetry_data = huawei_telemetry_pb2.Telemetry.FromString(i.data)
            print (telemetry_data)
```

Telemetry_CPU_Info()继承了 huawei_grpc_dialout_pb2_grpc 中 gRPCDataserviceServicer 方法，具体可以在 huawei_grpc_dialout_pb2_grpc.py 文件中查看。

```
class Telemetry_CPU_Info(huawei_grpc_dialout_pb2_grpc.gRPCDataserviceServicer):
```

子类继承父类的函数 dataPublish(self, request_iterator, context)。

设备 Telemetry 数据会循环持续推送。每次推送都用字段“#####start#####”标示。

```
for i in request_iterator:
            print ('############ start ############\n')
```

获取的的数据通过变量 telemetry_data 表示。注意此处需要使用 huawei_telemetry_pb2.Telemetry 类中的方法 FromString，并且只输入 request_iterator 中的 data 属性。

```
telemetry_data = huawei_telemetry_pb2.Telemetry.FromString(i.data)
print (telemetry_data)
```

输出 Telemetry_data 如下：

```
############ start ############

node_id_str: "CE1"
subscription_id_str: "Sub1"
sensor_path: "huawei-devm:devm/cpuInfos/cpuInfo"
collection_id: 260
```

```
collection_start_time: 1580278055740
msg_timestamp: 1580278055824
data_gpb {
  row {
    timestamp: 1580278055740
    content: "*\022\n\020\"\0011\010\201\200\204\010(\006\030Z0K\020\010"
  }
  row {
    timestamp: 1580278055740
    content: "*\023\n\021\"\00217\010\201\200\304\010(\006\030Z0K\020\010"
  }
}
collection_end_time: 1580278055740
current_period: 60000
except_desc: "OK"
product_name: "CE12800"
proto_path: "huawei_devm.Devm"
```

可以发现 Telemetry 完整数据包含 node_id、订阅 id、时间等参数。其中 data_gpb 中包含多个 row，每个 row 中包含 timestamp 和 content。这里我们发现 content 里的数据无法识别，需要进一步解码。

步骤 5 解码 Telemetry 数据

本例中真实需要获取的 CPU 信息包含在 content 字段无法识别，需要在对应的 proto 文件解码。

因为一次推送数据中包含多个 row，使用 for 循环读取处理。

```
for row_data in telemetry_data.data_gpb.row:
    print ('-----------------')
    print ('The proto path is :'+telemetry_data.proto_path)
    print ('-----------------')
```

查看每次数据对应的 proto_path，回显如下：

```
-----------------
The proto path is :huawei_devm.Devm
-----------------
```

每次根据不同的 proto_path 动态加载不同的模块。首先使用 split()方法将 huawei_devm.Devm 划分为 huawei_devm 和 Devm 两个字符串。

```
    module_name = telemetry_data.proto_path.split('.')[0]
    root_class = telemetry_data.proto_path.split('.')[1]

#动态加载 telemetry 获取数据的对应模块，本例中为
    decode_module = importlib.import_ module( module_name+'_pb2')
    print (decode_module)
```

使用 importlib.import_module 动态加载模块。并输出其内容。本例中模块为 huawei_devm_pb2。

```
<module 'huawei_devm_pb2' from 'D:\\10 Python Learning\\Telemetry\\huawei_devm_pb2.py'>
```

```
        #定义解码方法：getattr 获取动态加载的模块中的属性值，调用此属性的解码方法 FromString
                decode_func = getattr(decode_module,root_class).FromString
```

定义一个方法 decode_func，作用是实现 huawei_devm 数据的解码。

```
                print ('----------- content is -----------\n')
                # 将 row_data 中的 content 中的内容使用此方法解码，并输出
                print (decode_func(row_data.content))
                print ('----------- done -----------------')
```

最后将需要解码的 content 内容输入到此方法中。

实现了将无法识别的

```
data_gpb {
  row {
    timestamp: 1580278055740
    content: "*\022\n\020\"\0011\010\201\200\204\010(\006\030Z0K\020\010"
  }
  row {
    timestamp: 1580278055740
    content: "*\023\n\021\"\00217\010\201\200\304\010(\006\030Z0K\020\010"
  }
}
```

解码为：

```
cpuInfos {
  cpuInfo {
    entIndex: 16842753
    interval: 8
    ovloadThreshold: 90
    position: "1"
    systemCpuUsage: 6
    unovloadThreshold: 75
  }
}
```

与

```
cpuInfos {
  cpuInfo {
    entIndex: 17891329
    interval: 8
    ovloadThreshold: 90
    position: "17"
    systemCpuUsage: 6
    unovloadThreshold: 75
  }
}
```

分别对应两个 row 内的 content。

6.5 思考题

1. data_gpb 中为什么不传递完整数据，而需要解码?
2. 本例中获取数据和解码数据在一个函数中略显臃肿，能否有办法优化?

7 OPS 实验

7.1 实验说明

某公司现有一台 CE12800 设备，管理 IP 地址为 192.168.56.100。现在需要测试设备 OPS（Open Programmability System）功能，查询并清除设备的启动配置文件。

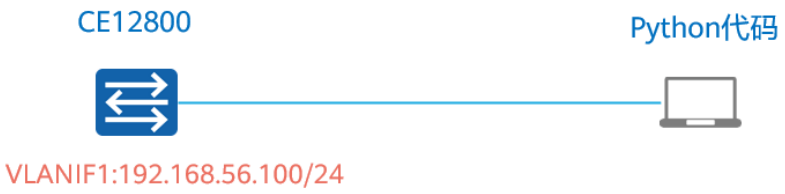
7.1.1 实验目标

- 了解 HTTP 常用操作。
- 掌握华为 OPS（Open Programmability System）开发能力。

7.1.2 实验环境准备

本实验需要额外准备 FTP 服务器/软件（本手册不涉及 FTP 服务器操作）。

使用模拟器或者真机准备实验环境。本实验本地 PC 运行 FTP 服务器（本例地址为 192.168.56.1），交换机为 FTP 客户端（本例 CE1 地址为 192.168.56.100）向服务器下载 Python 脚本。



1.配置 CE12800。

```
<HUAWEI>system-view immediately
Enter system view, return user view with return command.
[HUAWEI]sysname CE1
[CE1]interface Vlanif 1
[CE1-Vlanif1]ip add 192.168.56.100 24
[CE1-Vlanif1]quit
```

2.验证可达性。

登录本地 CMD 窗口测试本机到 CE1 的连通性。

```
C:\Users\XXX>ping 192.168.56.100
正在 Ping 192.168.56.100 具有 32 字节的数据:
```

```
来自 192.168.56.100 的回复: 字节=32 时间=3ms TTL=255
来自 192.168.56.100 的回复: 字节=32 时间=3ms TTL=255
来自 192.168.56.100 的回复: 字节=32 时间=2ms TTL=255
来自 192.168.56.100 的回复: 字节=32 时间=4ms TTL=255
192.168.56.100 的 Ping 统计信息:
    数据包: 已发送 = 4，已接收 = 4，丢失 = 0 (0% 丢失)，
往返行程的估计时间(以毫秒为单位):
    最短 = 2ms，最长 = 4ms，平均 = 3ms
```

3.测试 FTP 联通性

```
<CE1>ftp 192.168.56.1
Trying 192.168.56.1 ...
Press CTRL + K to abort
Connected to 192.168.56.1.
User(192.168.56.1:(none)):
```

FTP 测试通过。

7.2 配置思路

1. PC 上编写 OPS 脚本 demo.py。
2. 上传 Python 脚本到交换机。
3. 交换机运行 demo.py。

7.3 配置过程和完整代码

7.3.1 完整代码

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import traceback
import httplib
import string


# 定义调用 RESTful API 的类，该类中定义了一些方法来执行建立 HTTP 连接时的操作。该部分无需修改，用户可以直接使用。
# 该部分可以直接调用，用户不需要修改。
class OPSConnection(object):
    """Make an OPS connection instance."""

    # 初始化类，创建一个 HTTP 连接。
```

```
    def __init__(self, host, port=80):
        self.host = host
        self.port = port
        self.headers = {
            "Content-type": "text/xml",
            "Accept": "text/xml"
        }
        self.conn = None

    # 关闭 HTTP 连接。
    def close(self):
        """Close the connection"""
        self.conn.close()

    # 创建设备资源操作。
    def create(self, uri, req_data):
        """Create operation"""
        ret = self.rest_call("POST", uri, req_data)
        return ret

    # 删除设备资源操作。
    def delete(self, uri, req_data):
        """Delete operation"""
        ret = self.rest_call("DELETE", uri, req_data)
        return ret

    # 查询设备资源操作。
    def get(self, uri, req_data=None):
        """Get operation"""
        ret = self.rest_call("GET", uri, req_data)
        return ret

    # 修改设备资源操作。
    def set(self, uri, req_data):
        """Set operation"""
        ret = self.rest_call("PUT", uri, req_data)
        return ret

    # 类内部调用的方法。
    def rest_call(self, method, uri, req_data):
        """REST call"""
        print('|---------------------------------- request: ----------------------------------|')
        print('%s %s HTTP/1.1\n' % (method, uri))
        if req_data == None:
            body = ""
        else:
            body = req_data
```

```
            print(body)
        if self.conn:
            self.conn.close()
        self.conn = httplib.HTTPConnection(self.host, self.port)

        self.conn.request(method, uri, body, self.headers)
        response = self.conn.getresponse()
        response.status = httplib.OK    # stub code
        ret = (response.status, response.reason, response.read())
        print('|---------------------------------- response: ----------------------------------|')
        print('HTTP/1.1 %s %s\n\n%s' % ret)
        print('|-------------------------------------------------------------------------------|')
        return ret

def clear_startup_info(ops_conn):
    # 指定系统启动信息的 URI。URI 为 Resetful API 中定义的管理对象，不同的管理对象有不同的 URI。
    # 用户需要根据实际需求对 URI 进行修改，关于设备支持的 URI 可参考 RESTful API。
    uri = "/cfg/clearStartup"

    # 指定发送的请求内容。该部分内容与 URI 相对应，不同的 URI 对应不同的请求内容。
    # 用户需要根据实际使用的 URI 对请求内容进行修改，关于请求内容的格式可参考 RESTful API。
    req_data = \
        '''<?xml version="1.0" encoding="UTF-8"?>
        <clearStartup>
        </clearStartup>
        '''

    # 执行一个 POST 操作请求。uri 和 req_data 为请求 URI 和请求内容。ret 为请求是否成功的标识，
rsp_data 为请求执行后系统的响应数据，关于响应数据的格式可参考 RESTful API。

    # 用户可以根据实际需求对请求类型 get()进行修改，例如修改为 set()或者 create()。
    ret, _, rsp_data = ops_conn.create(uri, req_data)
    if ret != httplib.OK:
        return None

# 定义获取系统启动信息的函数。
def get_startup_info(ops_conn):
    # 指定系统启动信息的 URI。URI 为 Resetful API 中定义的管理对象，不同的管理对象有不同的 URI。
    # 用户需要根据实际需求对 URI 进行修改，关于设备支持的 URI 可参考 RESTful API。
    uri = "/cfg/startupInfos/startupInfo"

    # 指定发送的请求内容。该部分内容与 URI 相对应，不同的 URI 对应不同的请求内容。
    # 用户需要根据实际使用的 URI 对请求内容进行修改，关于请求内容的格式可参考 RESTful API。
    req_data = \
        '''<?xml version="1.0" encoding="UTF-8"?>
        <startupInfo>
```

```
            </startupInfo>
            '''

    # 执行一个 GET 操作请求。uri 和 req_data 为请求 URI 和请求内容。ret 为请求是否成功的标识，
rsp_data 为请求执行后系统的响应数据，关于响应数据的格式可参考 RESTful API。

    # 用户可以根据实际需求对请求类型 get()进行修改，例如修改为 set()或者 create()。
    ret, _, rsp_data = ops_conn.get(uri, req_data)
    if ret != httplib.OK:
        return None

    return rsp_data

    # main()函数定义脚本运行时需要执行的操作，用户可根据实际需求进行修改。


def main():
    """The main function."""

    # host 表示环路地址，当前 RESTful API 为设备内部调用，即取值为“localhost”。
    host = "localhost"
    try:
        # 建立 HTTP 连接。
        ops_conn = OPSConnection(host)
        # 调用获取系统启动信息的函数。
        rsp_data = get_startup_info(ops_conn)
        rsp_data = clear_startup_info(ops_conn)
        rsp_data = get_startup_info(ops_conn)
```

```
        # 关闭 HTTP 连接。
        ops_conn.close()
        return

    except:
        errinfo = traceback.format_exc()
        print(errinfo)
        return


if __name__ == "__main__":
    main()
```

7.3.2 上传代码

上述代码在 FTP 服务器存为 demo.py。

交换机作为 ftp 客户端下载此文件。

```
<CE1>ftp 192.168.56.1
```

```
Trying 192.168.56.1 ...
Press CTRL + K to abort
Connected to 192.168.56.1.
220
User(192.168.56.1:(none)):admin
Enter password:
230
[ftp]get demo.py
Warning: The file may not transfer correctly in ASCII mode.
213 5554
200 PORT                .
150 starting
/       0% [              ]
\       100% [**********]
226 Transfer complete.
```

成功下载文件。

7.3.3 运行结果

交换机有多种方法配置 OPS，本案例使用手动方式安装和运行 Python 脚本。

<CE1>ops install file demo.py

<CE1>ops run python demo.py

```
|---------------------------------- request: ----------------------------------|
GET /cfg/startupInfos/startupInfo HTTP/1.1

<?xml version="1.0" encoding="UTF-8"?>
          <startupInfo>
          </startupInfo>

|---------------------------------- response: ----------------------------------|
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply>
  <data>
    <cfg xmlns="http://www.huawei.com/netconf/vrp" format-version="1.0" content-
version="1.0">
      <startupInfos>
        <startupInfo>
          <position>17</position>
          <nextStartupFile>cfcard:/vrpcfg.cfg</nextStartupFile>
          <configedSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</con
figedSysSoft>
          <curSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</curSysSo
ft>
          <nextSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</nextSys
Soft>
```

```
            <curStartupFile>cfcard:/vrpcfg.cfg</curStartupFile>
            <curPatchFile>NULL</curPatchFile>
            <nextPatchFile>NULL</nextPatchFile>
            <boardInfo>101</boardInfo>
            <curPafFile>default</curPafFile>
            <nextPafFile>default</nextPafFile>
          </startupInfo>
        </startupInfos>
      </cfg>
  </data>
</rpc-reply>

|-------------------------------------------------------------------------|
|---------------------------------- request: ----------------------------------|
POST /cfg/clearStartup HTTP/1.1

<?xml version="1.0" encoding="UTF-8"?>
          <clearStartup>
          </clearStartup>

|---------------------------------- response: ----------------------------------|
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply>
  <ok/>
</rpc-reply>

|-------------------------------------------------------------------------|
|---------------------------------- request: ----------------------------------|
GET /cfg/startupInfos/startupInfo HTTP/1.1

<?xml version="1.0" encoding="UTF-8"?>
          <startupInfo>
          </startupInfo>

|---------------------------------- response: ----------------------------------|
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply>
  <data>
    <cfg xmlns="http://www.huawei.com/netconf/vrp" format-version="1.0" content-
version="1.0">
      <startupInfos>
        <startupInfo>
          <position>17</position>
          <nextStartupFile>NULL</nextStartupFile>
```

```
            <configedSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</con
figedSysSoft>
            <curSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</curSysSo
ft>
            <nextSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</nextSys
Soft>
            <curStartupFile>NULL</curStartupFile>
            <curPatchFile>NULL</curPatchFile>
            <nextPatchFile>NULL</nextPatchFile>
            <boardInfo>101</boardInfo>
            <curPafFile>default</curPafFile>
            <nextPafFile>default</nextPafFile>
          </startupInfo>
        </startupInfos>
      </cfg>
    </data>
</rpc-reply>

|---------------------------------------------------------------------------|
```

7.4 代码解析

步骤 1 导入模块

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import traceback
import httplib
import string
```

导入 OPS 功能需要使用的相关模块。因为此代码最终在交换机上运行，PC 无相关模块无需安装。

步骤 2 main 函数

根据代码执行顺序，首先解析 main 函数。

```
def main():
    """The main function."""

    # host 表示环路地址，当前 RESTful API 为设备内部调用，即取值为“localhost”。
    host = "localhost"
    try:
        # 建立 HTTP 连接。
        ops_conn = OPSConnection(host)
        # 调用获取系统启动信息的函数。
```

```
            rsp_data = get_startup_info(ops_conn)
            rsp_data = clear_startup_info(ops_conn)
            rsp_data = get_startup_info(ops_conn)
            # 关闭 HTTP 连接。
            ops_conn.close()
            return

        except:
            errinfo = traceback.format_exc()
            print(errinfo)
            return

if __name__ == "__main__":
    main()
```

main 函数首先定义本地为 host。交换机本地运行此脚本，将调用本地的 Restful 接口。

然后使用 try...except...捕获异常。其中首先调用 OPSConnection(host)类建立 HTTP 连接，其次先后调用 get_startup_info(ops_conn)函数获取当前设备配置信息，clear_startup_info(ops_conn)函数清除下一次设备启动配置，get_startup_info(ops_conn)函数再次获取当前设备配置信息验证。

最后调用 ops_conn.close()关闭 HTTP 连接。

步骤 3 OPSConnection()类

```
# 定义调用 RESTful API 的类，该类中定义了一些方法来执行建立 HTTP 连接时的操作。该部分无需修改，用户可以直接使用。
# 该部分可以直接调用，用户不需要修改。
class OPSConnection(object):
    """Make an OPS connection instance."""

    # 初始化类，创建一个 HTTP 连接。
    def __init__(self, host, port=80):
        self.host = host
        self.port = port
        self.headers = {
            "Content-type": "text/xml",
            "Accept": "text/xml"
        }
        self.conn = None

    # 关闭 HTTP 连接。
    def close(self):
        """Close the connection"""
        self.conn.close()

    # 创建设备资源操作。
    def create(self, uri, req_data):
```

```
        """Create operation"""
        ret = self.rest_call("POST", uri, req_data)
        return ret

    # 删除设备资源操作。
    def delete(self, uri, req_data):
        """Delete operation"""
        ret = self.rest_call("DELETE", uri, req_data)
        return ret

    # 查询设备资源操作。
    def get(self, uri, req_data=None):
        """Get operation"""
        ret = self.rest_call("GET", uri, req_data)
        return ret

    # 修改设备资源操作。
    def set(self, uri, req_data):
        """Set operation"""
        ret = self.rest_call("PUT", uri, req_data)
        return ret

    # 类内部调用的方法。
    def rest_call(self, method, uri, req_data):
        """REST call"""
        print('|---------------------------------- request: ----------------------------------|')
        print('%s %s HTTP/1.1\n' % (method, uri))
        if req_data == None:
            body = ""
        else:
            body = req_data
            print(body)
        if self.conn:
            self.conn.close()
        self.conn = httplib.HTTPConnection(self.host, self.port)

        self.conn.request(method, uri, body, self.headers)
        response = self.conn.getresponse()
        response.status = httplib.OK    # stub code
        ret = (response.status, response.reason, response.read())
        print('|---------------------------------- response: ----------------------------------|')
        print('HTTP/1.1 %s %s\n\n%s' % ret)
        print('|-------------------------------------------------------------------------------|')
        return ret
```

OPSConnection()为设备本地调用 RESTful API 的类。该类中定义了一些方法来执行建立 HTTP 连接时的操作（包括 GET、POST、DELETE、PUT 等操作）。该部分无需修改，用户可以直接使用。

步骤 4 get_startup_info()函数

此函数的作用是获取当前设备的启动信息。

```
def get_startup_info(ops_conn):
    # 指定系统启动信息的 URI。URI 为 Resetful API 中定义的管理对象，不同的管理对象有不同的 URI。
    # 用户需要根据实际需求对 URI 进行修改，关于设备支持的 URI 可参考 RESTful API。
    uri = "/cfg/startupInfos/startupInfo"
```

不同的操作对应不同的 URI，可以在《CloudEngine XXX RESTful API 参考》相应章节查询。例如本例对应内容为：

配置管理

系统启动信息

操作	URI	描述
GET	/cfg/startupInfos/startupInfo	获取系统启动信息。

- 请求示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<startupInfo>
</startupInfo>
```

本次将执行 GET 操作，所以 URI 填写为/cfg/startupInfos/startupInfo。

```
    # 指定发送的请求内容。该部分内容与 URI 相对应，不同的 URI 对应不同的请求内容。
    # 用户需要根据实际使用的 URI 对请求内容进行修改，关于请求内容的格式可参考 RESTful API。
    req_data = \
        '''<?xml version="1.0" encoding="UTF-8"?>
        <startupInfo>
        </startupInfo>
        '''
```

定义 req_data 为发送的请求信息。根据文档要求，填写请求如上。

```
    # 执行一个 GET 操作请求。uri 和 req_data 为请求 URI 和请求内容。ret 为请求是否成功的标识，
rsp_data 为请求执行后系统的响应数据，关于响应数据的格式可参考 RESTful API。

    # 用户可以根据实际需求对请求类型 get()进行修改，例如修改为 set()或者 create()。
    ret, _, rsp_data = ops_conn.get(uri, req_data)
    if ret != httplib.OK:
        return None

    return rsp_data
```

定义好 URI 和请求后，调用 ops_conn.get()方法对应 GET 操作。

主函数第一次调用步骤 4get_startup_info()函数会得到回显信息：

```
|---------------------------------- request: ----------------------------------|
GET /cfg/startupInfos/startupInfo HTTP/1.1

<?xml version="1.0" encoding="UTF-8"?>
          <startupInfo>
          </startupInfo>

|---------------------------------- response: ----------------------------------|
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply>
  <data>
    <cfg xmlns="http://www.huawei.com/netconf/vrp" format-version="1.0" content-
version="1.0">
      <startupInfos>
        <startupInfo>
          <position>17</position>
          <nextStartupFile>cfcard:/vrpcfg.cfg</nextStartupFile>
          <configedSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</con
figedSysSoft>
          <curSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</curSysSo
ft>
          <nextSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</nextSys
Soft>
          <curStartupFile>cfcard:/vrpcfg.cfg</curStartupFile>
          <curPatchFile>NULL</curPatchFile>
          <nextPatchFile>NULL</nextPatchFile>
          <boardInfo>101</boardInfo>
          <curPafFile>default</curPafFile>
          <nextPafFile>default</nextPafFile>
        </startupInfo>
      </startupInfos>
    </cfg>
  </data>
</rpc-reply>
|-----------------------------------------------------------------------------|
```

可以看到当前配置文件和下一次启动配置文件为 cfcard:/vrpcfg.cfg。

步骤 5 clear_startup_info()函数

此函数的作用是清除当前设备的配置文件。

```
def clear_startup_info(ops_conn):
    # 指定系统启动信息的 URI。URI 为 Resetful API 中定义的管理对象，不同的管理对象有不同的 URI。
    # 用户需要根据实际需求对 URI 进行修改，关于设备支持的 URI 可参考 RESTful API。
    uri = "/cfg/clearStartup"
```

同样参考文档，本次操作为清除下次启动文件的设置。所以对应的 URI 为"/cfg/clearStartup"。

下次启动配置文件

操作	URI	描述
POST	/cfg/setStartup	设置下次启动配置文件。
POST	/cfg/deleteStartup	删除下次启动配置文件。
POST	/cfg/clearStartup	取消下次启动文件的设置，系统本次启动文件和下次启动文件的设置将为空。

```
    # 指定发送的请求内容。该部分内容与 URI 相对应，不同的 URI 对应不同的请求内容。
    # 用户需要根据实际使用的 URI 对请求内容进行修改，关于请求内容的格式可参考 RESTful API。
    req_data = \
        '''<?xml version="1.0" encoding="UTF-8"?>
        <clearStartup>
        </clearStartup>
        '''
```

继续查询文档获得清除配置的请求格式。

3. 清除下次启动文件的设置：

- 请求示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<clearStartup>
</clearStartup>
```

```
    # 执行一个 POST 操作请求。uri 和 req_data 为请求 URI 和请求内容。ret 为请求是否成功的标识，
rsp_data 为请求执行后系统的响应数据，关于响应数据的格式可参考 RESTful API。

    # 用户可以根据实际需求对请求类型 get()进行修改，例如修改为 set()或者 create()。
    ret, _, rsp_data = ops_conn.create(uri, req_data)
    if ret != httplib.OK:
        return None
```

根据文档，清除配置对应的 HTTP 操作为 POST。查看 OPSConnection()里的方法，对应为 ops_conn.create。

主函数执行 clear_startup_info()函数会得到以下回显信息：

```
|---------------------------------- request: ----------------------------------|
POST /cfg/clearStartup HTTP/1.1

<?xml version="1.0" encoding="UTF-8"?>
        <clearStartup>
        </clearStartup>

|---------------------------------- response: ----------------------------------|
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8"?>
```

```
<rpc-reply>
  <ok/>
</rpc-reply>

|-------------------------------------------------------------------------------|
```

HTTP 请求头部是 POST 操作，内容为清除配置。HTTP 响应为成功。

最后再次执行 get_startup_info()函数以查看修改结果：

```
|---------------------------------- request: ----------------------------------|
GET /cfg/startupInfos/startupInfo HTTP/1.1

<?xml version="1.0" encoding="UTF-8"?>
          <startupInfo>
          </startupInfo>

|---------------------------------- response: ----------------------------------|
HTTP/1.1 200 OK

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply>
  <data>
    <cfg xmlns="http://www.huawei.com/netconf/vrp" format-version="1.0" content-
version="1.0">
      <startupInfos>
        <startupInfo>
          <position>17</position>
          <nextStartupFile>NULL</nextStartupFile>
          <configedSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</con
figedSysSoft>
          <curSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</curSysSo
ft>
          <nextSysSoft>cfcard:/VRPV200R005C10SPC607B607D0306_ce12800.cc</nextSys
Soft>
          <curStartupFile>NULL</curStartupFile>
          <curPatchFile>NULL</curPatchFile>
          <nextPatchFile>NULL</nextPatchFile>
          <boardInfo>101</boardInfo>
          <curPafFile>default</curPafFile>
          <nextPafFile>default</nextPafFile>
        </startupInfo>
      </startupInfos>
    </cfg>
  </data>
</rpc-reply>

|-------------------------------------------------------------------------------|
```

可以看到当前配置文件和下次启动配置文件为 NULL，修改成功。

7.5 思考题

交换机有个功能叫零配置开局 ZTP（Zero Touch Provisioning）。它的作用是新出厂或空配置设备上电启动时采用的一种自动加载版本文件（包括系统软件、配置文件、License 文件、补丁文件、自定义文件）的功能。请思考如何使用 OPS 实现 ZTP?

8 网络流量分析实验

8.1 实验说明

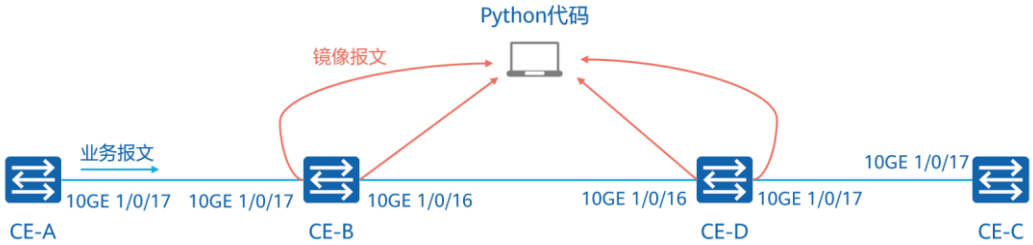
网络管理需求中要求识别网络中特定流量的转发路径。现某公司网络管理员计划将网络中的设备端口开启远程镜像功能，通过编写 Python 脚本，使用 Scapy 模块抓取镜像报文并分析，最后输出流量路径。

8.1.1 实验目标

- 实现网络设备远程镜像的配置。
- 掌握 Scapy 模块的基本使用。

8.1.2 实验环境准备

使用 CE12800 真机准备实验环境。组网信息如下图：业务报文从 CE-A 发出，达到 CE-C。在 CE-B 和 CE-D 交换机的 10GE 1/0/16 和 10GE 1/0/17 都开启远程镜像功能，将镜像报文发送到流量采集器（ Python 代码 ）。



1.安装远程镜像插件。

CE12800 远程镜像功能从 V200R001C00 版本起以插件化方式交付，系统软件中不包含上述功能。插件安装请参考《 CloudEngineXXX 插件操作指导-远程镜像 》文档。插件安装完成后，即可在 CE12800 使用远程镜像功能。

2.在 CE-B 配置远程镜像功能。

```
<CE-B>system-view immediately
Enter system view, return user view with return command.
[CE-B]observe-port 2 destination-ip 10.166.231.109 source-ip 10.154.186.102 erspan-id 2
[CE-B]interface 10GE 1/0/17
[CE-B-10GE1/0/17]port-mirroring observe-port 2 both
[CE-B-10GE1/0/17]interface 10GE 1/0/16
[CE-B-10GE1/0/16]port-mirroring observe-port 2 both
[CE-B-10GE1/0/16]quit
```

本例中 CE-B 的 IP 地址为 10.154.186.102，Python 代码所在采集器地址为 10.166.231.109。

3.在 CE-D 配置远程镜像功能。

```
<CE-D>system-view immediately
Enter system view, return user view with return command.
[CE-D]observe-port 3 destination-ip 10.166.231.109 source-ip 10.154.186.104 erspan-id 3
[CE-D]interface 10GE 1/0/17
[CE-D-10GE1/0/17]port-mirror observe-port 3 both
[CE-D-10GE1/0/17]interface 10GE 1/0/16
[CE-D-10GE1/0/16]port-mirror observe-port 3 both
[CE-D-10GE1/0/16]quit
```

本例中 CE-D 的 IP 地址为 10.154.186.104，Python 代码所在采集器地址为 10.166.231.109。

8.2 配置思路

1. PC 上编写 Python 脚本，使用 Scapy 抓包，分析报文路径并输出。
2. CE-A 上使用 ping 及 telnet CE-C 来模拟业务报文。
3. CE-A ping 或 telnet CE-C 的同时，在 PC 上执行脚本。

8.3 操作过程和完整代码

8.3.1 完整代码

```
# -*- coding: utf-8 -*-

from scapy.all import *
from scapy.contrib.erspan import *
import re, paramiko

# 创建字典，存放 CE-B 和 CE-D 的组网信息。
interfaceConnect_CE_B = {
    "10GE1/0/17": {"CE-A": "10GE1/0/17"},
    "10GE1/0/14": {"CE-C": "10GE1/0/14"},
    "10GE1/0/16": {"CE-D": "10GE1/0/16"},
}

interfaceConnect_CE_D = {
    "10GE1/0/16": {"CE-B": "10GE1/0/16"},
    "10GE1/0/17": {"CE-C": "10GE1/0/17"},
}

# 创建类 NetDevice，用于抽象一个设备，获取其相关信息
class NetDevice:
```

```
def __init__(self, deviceName, deviceIp, username, password, interfaceConnect):
    self.deviceName = deviceName
    self.deviceIp = deviceIp
    self.username = username
    self.password = password
    self.interfaces = []
    self.interfacesConnect = {}
    self.macUnderInterface = {}
    self.interfaceMac = {}
    try:
        ssh = paramiko.client.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(hostname=deviceIp, username=username, password=password)
        cli = ssh.invoke_shell()
        cli.send('N\n')
        time.sleep(0.5)
        # 暂时关闭分屏，使命令行回显可以一屏输出
        cli.send('screen-length 0 temporary\n')
        time.sleep(1)
    except:
        print("SSH 登陆设备{}出现异常!".format(deviceIp))
        raise

    self.get_interfaces_info(cli)
    self.get_mac_under_interface(cli)
    self.interfacesConnect = interfaceConnect
    ssh.close()

# ********************************************************
# 作用：获取设备所有接口及其 mac 地址，并分别赋值到 self.interfaces
#       和 self.interfaceMac
# 参数：
#    cli —— ssh shell 对象
# ********************************************************
def get_interfaces_info(self, cli):
    cli.send('display interface\n')
    time.sleep(15)
    interfaces_info = cli.recv(999999).decode()
    results = re.findall(r'^(\S+) current state.*\(ifindex', interfaces_info, re.M)
    if len(results) > 0:
        self.interfaces = results
    else:
        print("没有获取到接口！")

    interface_mac = ["", ""]
    lines = interfaces_info.splitlines()
```

```
        for i in lines:
            result1 = re.search(r'^(\S+) current state.*\(ifindex', i)
            if result1:
                interface_mac[0] = result1.group(1)

            result2 = re.search(r'Hardware address is (\S{4}-\S{4}-\S{4})', i)
            if result2:
                interface_mac[1] = result2.group(1)
                self.interfaceMac[interface_mac[0]] = interface_mac[1]

    # ****************************************************************
    # 作用：获取从设备接口学习到的 mac 地址，并将其赋值到 self.macUnderInterface
    # 参数：
    #    cli —— ssh shell 对象
    # ****************************************************************
    def get_mac_under_interface(self, cli):
        cli.send('display mac-address\n')
        time.sleep(3)
        mac_address_info = cli.recv(999999).decode()
        cli.send('display arp\n')
        time.sleep(3)
        arp_info = cli.recv(999999).decode()

        mac_interface = re.findall(r'^(\S{4}-\S{4}-\S{4}) \S+ +(\S+) ', mac_address_info, re.M)
        if len(mac_interface) > 0:
            for i in mac_interface:
                if i[1] not in self.macUnderInterface.keys():
                    self.macUnderInterface[i[1]] = []
                    self.macUnderInterface[i[1]].append(i[0])
                else:
                    if i[0] not in self.macUnderInterface[i[1]]:
                        self.macUnderInterface[i[1]].append(i[0])
        else:
            print("MAC 地址表没有表项！")

        arp_interface = re.findall(r' (\S{4}-\S{4}-\S{4}) .*D.* +(\S+)', arp_info, re.M)
        if len(arp_interface) > 0:
            for i in arp_interface:
                if i[1] not in self.macUnderInterface.keys():
                    self.macUnderInterface[i[1]] = []
                    self.macUnderInterface[i[1]].append(i[0])
                else:
                    if i[0] not in self.macUnderInterface[i[1]]:
                        self.macUnderInterface[i[1]].append(i[0])
        else:
            print("ARP 表项没有动态表项。")
```

```
    # ******************************************************
    # 作用：判断报文是否发送到设备
    # 参数：
    #    packet_info —— 报文信息
    # 返回值：
    #    True/False
    # ******************************************************
    def is_to_device(self, packet_info):
        for key in self.interfaceMac.keys():
            # MAC 地址可能有 08e8-4f6e-0516, 08:e8:4f:6e:05:16 这两种格式，对其做一下转换再进行比
较
            if re.sub(r"[-:]", "", packet_info.get("macDst")) == re.sub(r"[-:]", "",
self.interfaceMac.get(key)):
                return True
        return False

    # ******************************************************
    # 作用：判断报文是否由本设备发送出去
    # 参数：
    #    packet_info —— 报文信息
    # 返回值：
    #    True/False
    # ******************************************************
    def is_from_device(self, packet_info):
        for key in self.interfaceMac.keys():
            # MAC 地址可能有 08e8-4f6e-0516, 08:e8:4f:6e:05:16 这两种格式，对其做一下转换再进行比
较
            if re.sub(r"[-:]", "", packet_info.get("macSrc")) == re.sub(r"[-:]", "",
self.interfaceMac.get(key)):
                return True
        return False

    # ******************************************************
    # 作用：获取报文入接口及对端设备接口，并返回该路径段
    # 参数：
    #    packet_info —— 报文信息
    # 返回值：
    #    packet_path
    #    [{"CE1": "interface1"},
    #     {"CE2": "interface2"}]
    #    表示报文从对端设备 CE1 的 interface1 进入本设备 CE2 的 interface2
    # ******************************************************
    def in_device_path(self, packet_info):
        # 报文从对端设备发送的路径信息
        packet_path_out = {}
        # 报文从本端设备接收的路径信息
```

```
        packet_path_in = {}
        interface = ""
        for key in self.macUnderInterface.keys():
            macs = self.macUnderInterface.get(key)
            for i in range(len(macs)):
                macs[i] = re.sub(r"[-:]", "", macs[i])

            if re.sub(r"[-:]", "", packet_info.get("macSrc")) in macs:
                interface = key
                break

        if interface == "":
            raise Exception('报文在设备上找不到入接口，报文信息：\n{}'.format(packet_info))

        if interface not in self.interfacesConnect:
            raise Exception('设备{}接口没有对端设备及对端接口信息'.format(interface))

        for key, value in self.interfacesConnect.get(interface).items():
            packet_path_out[key] = value

        packet_path_in[self.deviceName] = interface
        return [packet_path_out, packet_path_in]

    # ********************************************************
    # 作用：获取报文出接口及对端设备接口，并返回该路径段
    # 参数：
    #    packet_info —— 报文信息
    # 返回值：
    #    packet_path
    #    [{"CE1": "interface1"},
    #     {"CE2": "interface2"}]
    #    表示报文从本端设备 CE1 的 interface1 发出到对端设备 CE2 的 interface2
    # ********************************************************
    def out_device_path(self, packet_info):
        # 报文从本端设备发送的路径信息
        packet_path_out = {}
        # 报文在对端设备接收的路径信息
        packet_path_in = {}
        interface = ""
        for key in self.macUnderInterface.keys():
            macs = self.macUnderInterface.get(key)
            for i in range(len(macs)):
                macs[i] = re.sub(r"[-:]", "", macs[i])

            if re.sub(r"[-:]", "", packet_info.get("macDst")) in macs:
                interface = key
                break
```

```
        if interface == "":
            raise Exception('报文在设备上找不到出接口，报文信息：\n{}'.format(packet_info))

        if interface not in self.interfacesConnect:
            raise Exception('设备{}接口没有对端设备及对端接口信息'.format(interface))

        for key, value in self.interfacesConnect.get(interface).items():
            packet_path_in[key] = value

        packet_path_out[self.deviceName] = interface
        return [packet_path_out, packet_path_in]

# ******************************************************
# 作用：获取需要计算路径流量相关镜像报文
# 参数：
#    flwo_info —— 需要计算路径的流量信息
#    mirror_packets —— 所有镜像报文
# 返回值：
#    flow_packets —— 与需要计算路径流量相关的镜像报文
# ******************************************************
def get_flow_packets(flow_info, mirror_packets):
    flow_packets = []
    for packet in mirror_packets:
        if packet["ipSrc"] != flow_info["ipSrc"]:
            continue
        elif packet["ipDst"] != flow_info["ipDst"]:
            continue
        elif packet["ipProto"] != flow_info["ipProto"]:
            continue

        if flow_info["ipProto"] == "icmp":
            flow_packets.append(packet)
            continue
        elif flow_info["ipProto"] == "tcp" or flow_info["ipProto"] == "udp":
            if flow_info["portSrc"] != "any" and packet["portSrc"] != flow_info["portSrc"]:
                continue
            elif flow_info["portDst"] != "any" and packet["portDst"] != flow_info["portDst"]:
                continue
            else:
                flow_packets.append(packet)
    return flow_packets

# ******************************************************
# 作用：判断两个路径段是否相等
# 参数：
```

```
#     path1 —— 路径段 1
#     path2 —— 路径段 2
# 返回值:
#     True/False
# ********************************************************
def is_equal_path(path1, path2):
    for i in range(len(path1)):
        if path1[i] != path2[i]:
            return False
    return True

# ********************************************************
# 作用：将每一个路径段整合拼凑出完整报文路径
# 参数:
#     path_section —— 报文经过的每个远程镜像设备的路径段组成的 list
# 返回值:
#     path —— 报文路径
# ********************************************************
def integrate_path(path_section):
    # 去除 path_section 中的重复路径段
    path_section_norepetition = []
    path_section_norepetition.append(path_section[0])
    for k in range(len(path_section)):
        flag = False
        for h in path_section_norepetition:
            if is_equal_path(h, path_section[k]):
                flag = True
                break
        if not flag:
            path_section_norepetition.append(path_section[k])
    path = []
    path.append(path_section_norepetition[0][0])
    path.append(path_section_norepetition[0][1])
    del path_section_norepetition[0]
    while len(path_section_norepetition) > 0:
        last_device = list(path[len(path)-1].keys())[0]
        for i in range(len(path_section_norepetition)):
            if last_device in path_section_norepetition[i][0]:
                path.append(path_section_norepetition[i][0])
                path.append(path_section_norepetition[i][1])
                del path_section_norepetition[i]
                break

        if len(path_section_norepetition) == 0:
            break

        first_device = list(path[0].keys())[0]
```

```
            for j in range(len(path_section_norepetition)):
                if first_device in path_section_norepetition[j][1]:
                    path.insert(0, path_section_norepetition[j][1])
                    path.insert(0, path_section_norepetition[j][0])
                    del path_section_norepetition[j]
                    break
        return path

# ********************************************************
# 作用：将报文路径打印出来
# 参数：
#     path —— 包含报文完整路径的 list
# 返回值：
#     None
# ********************************************************
def print_path(path):
        path_str = ""
        i = 0
        while i < len(path):
            if i == 0:
                for key, value in path[i].items():
                    path_str += key + "(" + value + ")"
            elif i == len(path) - 1:
                for key, value in path[i].items():
                    path_str += " ---> (" + value + ")" + key
            else:
                for key, value in path[i].items():
                    path_str += " ---> (" + value + ")" + key
                if list(path[i].keys())[0] in path[i+1]:
                    for key, value in path[i+1].items():
                        path_str += "(" + value + ")"
                    i += 1
            i += 1
        print(path_str)

# ********************************************************
# 作用：计算并打印报文路径
# 参数：
#     flwo_info —— 需要计算路径的流量信息
#     mirror_packets —— 所有镜像报文
#     devices —— 所有开启远程镜像的设备
# 返回值：
#     None
# ********************************************************
def path_calc(flow_info, mirror_packets, devices):
        # 用于保存每一段报文路径
```

```
    path_section = []
    flow_packets = get_flow_packets(flow_info, mirror_packets)
    for packet in flow_packets:
        for device in devices:
            if device.is_to_device(packet):
                path = device.in_device_path(packet)
                path_section.append(path)
            elif device.is_from_device(packet):
                path = device.out_device_path(packet)
                path_section.append(path)
    path = integrate_path(path_section)
    print_path(path)

# ******************************************************
# 作用：提示用户输入需要计算路径的流量的五元组信息
# 参数：
#    None
# 返回值：
#    flwo_info —— 流量五元组信息，格式例如：
#    flow_info = {
#        "ipSrc": "10.1.12.2",
#        "ipDst": "192.168.2.2",
#        "ipProto": "tcp",
#        "portSrc": 23,
#        "portDst": "any",
#    }
# ******************************************************
def input_flow_info():
    flow_info = {}                  # 存放需要进行路径计算的流量五元组信息
    input_str = input("请输入要计算路径的流量五元组信息(源 IP,目的 IP,协议,源端口号,目的端口号):")
    input_list = input_str.split(",")
    flow_info["ipSrc"] = input_list[0].strip()
    flow_info["ipDst"] = input_list[1].strip()
    flow_info["ipProto"] = input_list[2].strip()
    flow_info["portSrc"] = ""
    flow_info["portDst"] = ""
    if flow_info["ipProto"] == "tcp" or flow_info["ipProto"] == "udp":
        portSrc = input_list[3].strip()
        if portSrc != "any":
            flow_info["portSrc"] = int(portSrc)
        else:
            flow_info["portSrc"] = "any"

        portDst = input_list[4].strip()
        if portDst != "any":
            flow_info["portDst"] = int(portDst)
        else:
```

```
                flow_info["portDst"] = "any"
    return flow_info

# ****************************************************
# 作用：使用 scapy 模块抓包，并提取被镜像的业务报文信息
# 参数：
#    None
# 返回值：
#    packInfoList -- 列表，存放被镜像的业务报文信息
# ****************************************************
def packets_capture():
    # 使用 sniff 获取镜像流量。sniff 的 iface 参数要根据实际进行抓包的接口修改
    mirrorPackets = sniff(filter = 'ip proto 47', iface = 'Realtek 8822CE Wireless LAN 802.11ac PCI-E NIC',
timeout = 30)

    # 定义变量 packInfoList，用于存放被镜像的业务报文信息。
    packInfoList = []
    grePackets = []
    for packet in mirrorPackets:
        if packet.haslayer(GRE):
            # 判断 GRE 乘客协议是否是 ERSPAN
            if packet[GRE].proto == 0x88be:
                packInfo = {
                    "macSrc": "",
                    "macDst": "",
                    "ipSrc": "",
                    "ipDst": "",
                    "ipProto": "",
                    "portSrc": "",
                    "portDst": "",
                }
                erspanPack = packet[GRE][1]                    # 获取 GRE 内层的 ERSPAN 报文
                etherPack = erspanPack[1]                      # 获取 ERSPAN 报文内层的 Ethernet 报
文
                macSrc = etherPack.src
                macDst = etherPack.dst
                if etherPack.type != 0x0800:
                    continue
                ipPack = etherPack[1]                          # 获取 Ethernet 报文上层的 IP 报文
                ipSrc = ipPack.src
                ipDst = ipPack.dst
                ipProto = ""
                # 如果是 icmp 报文
                if ipPack.proto == 1:
                    ipProto = "icmp"
                elif ipPack.proto == 6:
                    ipProto = "tcp"
```

```
                    packInfo["portDst"] = ipPack[1].dport
                    packInfo["portSrc"] = ipPack[1].sport
                else:
                    # 这边先只处理 icmp 报文和 tcp 报文，后面再增加 udp
                    continue
                packInfo["macSrc"] = macSrc
                packInfo["macDst"] = macDst
                packInfo["ipSrc"] = ipSrc
                packInfo["ipDst"] = ipDst
                packInfo["ipProto"] = ipProto
                if packInfo not in packInfoList:
                    packInfoList.append(packInfo)
    return packInfoList

# 主函数
def main():
    flow_info = input_flow_info()
    packInfoList = packets_capture()
    devices = []
    CE_B = NetDevice("CE-B", "10.154.186.102", "python", "Huawei12#$", interfaceConnect_CE_B)
    CE_D = NetDevice("CE-D", "10.154.186.104", "python", "Huawei@1234", interfaceConnect_CE_D)
    devices.append(CE_B)
    devices.append(CE_D)
    path_calc(flow_info, packInfoList, devices)

if __name__ == "__main__":
    main()
```

8.3.2 操作步骤

本实验使用 ping 和 telnet 两种方式模拟业务报文。CE-A 10GE 1/0/17 接口 IP 地址为 192.168.2.2，CE-C 10GE 1/0/17 接口 IP 地址为 10.1.14.1。

1. CE-A ping CE-C，同时在本地 PC 运行脚本。

2. CE-A telnet CE-C，同时在本地 PC 运行脚本。

8.3.3 运行结果

1. 从 CE-A ping CE-C 流量路径。

icmp echo request 报文路径：

```
C:\Users\Anaconda3\python.exe D:/pyworkspace/scapy_expriment/mirrorAnalyse.py
请输入要计算路径的流量五元组信息(源 IP,目的 IP,协议,源端口号,目的端口号):192.168.2.2, 10.1.14.1, icmp
CE-A(10GE1/0/17) ---> (10GE1/0/17)CE-B(10GE1/0/16) ---> (10GE1/0/16)CE-D(10GE1/0/17) --->
(10GE1/0/17)CE-C

Process finished with exit code 0
```

icmp echo relpy 报文路径：

```
C:\Users\Anaconda3\python.exe D:/pyworkspace/scapy_expriment/mirrorAnalyse.py
请输入要计算路径的流量五元组信息(源 IP,目的 IP,协议,源端口号,目的端口号):10.1.14.1, 192.168.2.2, icmp
CE-C(10GE1/0/17) ---> (10GE1/0/17)CE-D(10GE1/0/16) ---> (10GE1/0/16)CE-B(10GE1/0/17) --->
(10GE1/0/17)CE-A

Process finished with exit code 0
```

2. 从 CE-A telnet CE-C 流量路径。

client 到 server 报文路径：

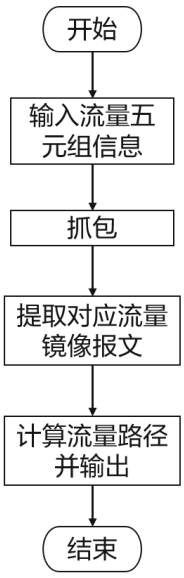
```
C:\Users\Anaconda3\python.exe D:/pyworkspace/scapy_expriment/mirrorAnalyse.py
请输入要计算路径的流量五元组信息(源 IP,目的 IP,协议,源端口号,目的端口号):192.168.2.2, 10.1.14.1, tcp,
any, 23
CE-A(10GE1/0/17) ---> (10GE1/0/17)CE-B(10GE1/0/16) ---> (10GE1/0/16)CE-D(10GE1/0/17) --->
(10GE1/0/17)CE-C

Process finished with exit code 0
```

server 到 client 报文路径：

```
C:\Users\Anaconda3\python.exe D:/pyworkspace/scapy_expriment/mirrorAnalyse.py
请输入要计算路径的流量五元组信息(源 IP,目的 IP,协议,源端口号,目的端口号):10.1.14.1, 192.168.2.2, tcp, 23,
any
CE-C(10GE1/0/17) ---> (10GE1/0/17)CE-D(10GE1/0/16) ---> (10GE1/0/16)CE-B(10GE1/0/17) --->
(10GE1/0/17)CE-A

Process finished with exit code 0
```

8.4 代码解析

脚本整体思路：



步骤 1 导入模块

```
from scapy.all import *
from scapy.contrib.erspan import *
import re, paramiko
```

导入脚本所需模块，脚本中需要 ssh 登录网络设备，采集并提取相关信息。因此，导入 re，paramiko 模块。

步骤 2 填写组网信息

```
# 创建字典，存放 CE-B 和 CE-D 的组网信息。
interfaceConnect_CE_B = {                                    # 表示 CE-B 设备的 10GE1/0/17 接口对端是
    "10GE1/0/17": {"CE-A": "10GE1/0/17"},                    # CE-A 设备的 10GE1/0/17 接口，以此类推。
    "10GE1/0/14": {"CE-C": "10GE1/0/14"},
    "10GE1/0/16": {"CE-D": "10GE1/0/16"},
}

interfaceConnect_CE_D = {
    "10GE1/0/16": {"CE-B": "10GE1/0/16"},
    "10GE1/0/17": {"CE-C": "10GE1/0/17"},
}
```

网络中设备的组网信息需要提前填写到脚本中，计算流量路径时需使用。

步骤 3 创建 NetDevice 类

NetDevice 类中定义了 6 个设备相关的方法，分别是 get_interfaces_info，get_mac_under_interface，is_to_device，is_from_device，in_device_path 和 out_device_path。

1. NetDevice 类。用于对网络设备进行抽象。

```
# 创建类 NetDevice，用于抽象一个设备，获取其相关信息
class NetDevice:

    def __init__(self, deviceName, deviceIp, username, password, interfaceConnect):
        self.deviceName = deviceName
        self.deviceIp = deviceIp
        self.username = username
        self.password = password
        self.interfaces = []
        self.interfacesConnect = {}
        self.macUnderInterface = {}
        self.interfaceMac = {}
        try:
            ssh = paramiko.client.SSHClient()
            ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
            ssh.connect(hostname=deviceIp, username=username, password=password)
            cli = ssh.invoke_shell()
            cli.send('N\n')
            time.sleep(0.5)
```

```
            # 暂时关闭分屏，使命令行回显可以一屏输出
            cli.send('screen-length 0 temporary\n')
            time.sleep(1)
        except:
            print("SSH 登陆设备{}出现异常!".format(deviceIp))
            raise

        self.get_interfaces_info(cli)
        self.get_mac_under_interface(cli)
        self.interfacesConnect = interfaceConnect
        ssh.close()
```

2. get_interfaces_info 方法。该方法用于获取网络设备接口的 Mac 地址。

```
    # ********************************************************
    # 作用：获取设备所有接口及其 mac 地址，并分别赋值到 self.interfaces
    #       和 self.interfaceMac
    # 参数：
    #   cli —— ssh shell 对象
    # ********************************************************
    def get_interfaces_info(self, cli):
        cli.send('display interface\n')
        time.sleep(15)
        interfaces_info = cli.recv(999999).decode()
        results = re.findall(r'^(\S+) current state.*\(ifindex', interfaces_info, re.M)
        if len(results) > 0:
            self.interfaces = results
        else:
            print("没有获取到接口！")

        interface_mac = ["", ""]
        lines = interfaces_info.splitlines()
        for i in lines:
            result1 = re.search(r'^(\S+) current state.*\(ifindex', i)
            if result1:
                interface_mac[0] = result1.group(1)

            result2 = re.search(r'Hardware address is (\S{4}-\S{4}-\S{4})', i)
            if result2:
                interface_mac[1] = result2.group(1)
                self.interfaceMac[interface_mac[0]] = interface_mac[1]
```

3. get_mac_under_interface 方法。该方法用于获取从网络设备接口学习到的 Mac 地址。

```
    # ************************************************************
    # 作用：获取从设备接口学习到的 mac 地址，并将其赋值到 self.macUnderInterface
    # 参数：
    #   cli —— ssh shell 对象
    # ************************************************************
```

```
    def get_mac_under_interface(self, cli):
        cli.send('display mac-address\n')
        time.sleep(3)
        mac_address_info = cli.recv(999999).decode()
        cli.send('display arp\n')
        time.sleep(3)
        arp_info = cli.recv(999999).decode()

        mac_interface = re.findall(r'^(\S{4}-\S{4}-\S{4}) \S+ +(\S+) ', mac_address_info, re.M)
        if len(mac_interface) > 0:
            for i in mac_interface:
                if i[1] not in self.macUnderInterface.keys():
                    self.macUnderInterface[i[1]] = []
                    self.macUnderInterface[i[1]].append(i[0])
                else:
                    if i[0] not in self.macUnderInterface[i[1]]:
                        self.macUnderInterface[i[1]].append(i[0])
        else:
            print("MAC 地址表没有表项！")

        arp_interface = re.findall(r' (\S{4}-\S{4}-\S{4}) .*D.* +(\S+)', arp_info, re.M)
        if len(arp_interface) > 0:
            for i in arp_interface:
                if i[1] not in self.macUnderInterface.keys():
                    self.macUnderInterface[i[1]] = []
                    self.macUnderInterface[i[1]].append(i[0])
                else:
                    if i[0] not in self.macUnderInterface[i[1]]:
                        self.macUnderInterface[i[1]].append(i[0])
        else:
            print("ARP 表项没有动态表项。")
```

4. is_to_device 方法。该方法使用之前方法采集的信息判断业务报文是否是发送到本网络设备的。若业务报文的目的 MAC 地址是属于本网络设备的 MAC 地址，则认为该报文是发送到本网络设备的。

```
    # ******************************************************
    # 作用：判断报文是否发送到设备
    # 参数：
    #    packet_info —— 报文信息
    # 返回值：
    #    True/False
    # ******************************************************
    def is_to_device(self, packet_info):
        for key in self.interfaceMac.keys():
            # MAC 地址可能有 08e8-4f6e-0516, 08:e8:4f:6e:05:16 这两种格式，对其做一下转换再进行比
较
```

```
                if re.sub(r"[-:]", "", packet_info.get("macDst")) == re.sub(r"[-:]", "",
self.interfaceMac.get(key)):
                    return True
        return False
```

5. is_from_device 方法。该方法使用之前方法采集的信息判断业务报文是否是从本网络设备发出的。若业务报文的源 MAC 地址是属于本网络设备的 MAC 地址，则认为该报文是从本网络设备发出的。

```
    # ********************************************************
    # 作用：判断报文是否由本设备发送出去
    # 参数：
    #    packet_info —— 报文信息
    # 返回值：
    #    True/False
    # ********************************************************
    def is_from_device(self, packet_info):
        for key in self.interfaceMac.keys():
            # MAC 地址可能有 08e8-4f6e-0516, 08:e8:4f:6e:05:16 这两种格式，对其做一下转换再进行比
较
            if re.sub(r"[-:]", "", packet_info.get("macSrc")) == re.sub(r"[-:]", "",
self.interfaceMac.get(key)):
                return True
        return False
```

6. in_device_path 方法。该方法会计算业务报文发送到本网络设备的路径，原理是通过对源 MAC 地址查找 MAC 地址表和 ARP 表确定该报文是从哪个接口进入本网络设备的，再根据组网信息确定对端设备及接口。如本网络设备为 CE2，业务报文是从设备 CE1 的 GE 1/0/0 发送到 CE2 的 GE1/0/2，则 in_device_path 会计算出路径 CE1(GE1/0/0) ---> (GE1/0/2)CE2。

```
    # ********************************************************
    # 作用：获取报文入接口及对端设备接口，并返回该路径段
    # 参数：
    #    packet_info —— 报文信息
    # 返回值：
    #    packet_path
    #    [{"CE1": "interface1"},
    #     {"CE2": "interface2"}]
    #    表示报文从对端设备 CE1 的 interface1 进入本设备 CE2 的 interface2
    # ********************************************************
    def in_device_path(self, packet_info):
        # 报文从对端设备发送的路径信息
        packet_path_out = {}
        # 报文从本端设备接收的路径信息
        packet_path_in = {}
        interface = ""
        for key in self.macUnderInterface.keys():
            macs = self.macUnderInterface.get(key)
```

```
            for i in range(len(macs)):
                macs[i] = re.sub(r"[-:]", "", macs[i])

            if re.sub(r"[-:]", "", packet_info.get("macSrc")) in macs:
                interface = key
                break

        if interface == "":
            raise Exception('报文在设备上找不到入接口，报文信息：\n{}'.format(packet_info))

        if interface not in self.interfacesConnect:
            raise Exception('设备{}接口没有对端设备及对端接口信息'.format(interface))

        for key, value in self.interfacesConnect.get(interface).items():
            packet_path_out[key] = value

        packet_path_in[self.deviceName] = interface
        return [packet_path_out, packet_path_in]
```

7. out_device_path 方法。该方法会计算从本网络设备发出的业务报文路径，原理是通过对目的 MAC 地址查找 MAC 地址表和 ARP 表确定该报文是从本网络设备哪个接口发出的，再根据组网信息确定对端设备及接口。如本网络设备为 CE1，业务报文是从设备 CE1 的 GE 1/0/0 发送到 CE2 的 GE1/0/2，则 out_device_path 会计算出路径 CE1(GE1/0/0) ---> (GE1/0/2)CE2。

```
    # ************************************************************
    # 作用：获取报文出接口及对端设备接口，并返回该路径段
    # 参数：
    #    packet_info —— 报文信息
    # 返回值：
    #    packet_path
    #    [{"CE1": "interface1"},
    #     {"CE2": "interface2"}]
    #    表示报文从本端设备 CE1 的 interface1 发出到对端设备 CE2 的 interface2
    # ************************************************************
    def out_device_path(self, packet_info):
        # 报文从本端设备发送的路径信息
        packet_path_out = {}
        # 报文在对端设备接收的路径信息
        packet_path_in = {}
        interface = ""
        for key in self.macUnderInterface.keys():
            macs = self.macUnderInterface.get(key)
            for i in range(len(macs)):
                macs[i] = re.sub(r"[-:]", "", macs[i])

            if re.sub(r"[-:]", "", packet_info.get("macDst")) in macs:
```

```
                    interface = key
                    break

        if interface == "":
            raise Exception('报文在设备上找不到出接口，报文信息：\n{}'.format(packet_info))

        if interface not in self.interfacesConnect:
            raise Exception('设备{}接口没有对端设备及对端接口信息'.format(interface))

        for key, value in self.interfacesConnect.get(interface).items():
            packet_path_in[key] = value

        packet_path_out[self.deviceName] = interface
        return [packet_path_out, packet_path_in]
```

步骤 4 输入流量五元组信息

input_flow_info 方法提示用户输入需要计算路径的流量的五元组信息，并存放到 flow_info 变量并返回。

```
# ****************************************************
# 作用：提示用户输入需要计算路径的流量的五元组信息
# 参数：
#    None
# 返回值：
#    flwo_info —— 流量五元组信息，格式例如：
#    flow_info = {
#          "ipSrc": "10.1.12.2",
#          "ipDst": "192.168.2.2",
#          "ipProto": "tcp",
#          "portSrc": 23,
#          "portDst": "any",
#    }
# ****************************************************
def input_flow_info():
    flow_info = {}                  # 存放需要进行路径计算的流量五元组信息
    input_str = input("请输入要计算路径的流量五元组信息(源 IP,目的 IP,协议,源端口号,目的端口号):")
    input_list = input_str.split(",")
    flow_info["ipSrc"] = input_list[0].strip()
    flow_info["ipDst"] = input_list[1].strip()
    flow_info["ipProto"] = input_list[2].strip()
    flow_info["portSrc"] = ""
    flow_info["portDst"] = ""
    if flow_info["ipProto"] == "tcp" or flow_info["ipProto"] == "udp":
        portSrc = input_list[3].strip()
        if portSrc != "any":
            flow_info["portSrc"] = int(portSrc)
        else:
```

```
            flow_info["portSrc"] = "any"

        portDst = input_list[4].strip()
        if portDst != "any":
            flow_info["portDst"] = int(portDst)
        else:
            flow_info["portDst"] = "any"
    return flow_info
```

步骤 5 Scapy 抓包

packets_capture 方法使用 Scapy 模块的 sniff 函数抓取镜像报文，并提取其封装的业务报文源目 MAC 地址，源目 IP 地址，协议，源目端口号信息，存放到 packInfoList 变量并返回。

更多 scapy 相关用法请参考官方文档，https://scapy.readthedocs.io/en/latest/introduction.html。

```
# ******************************************************
# 作用：使用 scapy 模块抓包，并提取被镜像的业务报文信息
# 参数：
#    None
# 返回值：
#    packInfoList -- 列表，存放被镜像的业务报文信息
# ******************************************************
def packets_capture():
    # 使用 sniff 获取镜像流量。sniff 的 iface 参数要根据实际进行抓包的接口修改
    mirrorPackets = sniff(filter = 'ip proto 47', iface = 'Realtek 8822CE Wireless LAN 802.11ac PCI-E NIC', timeout = 30)

    # 定义变量 packInfoList，用于存放被镜像的业务报文信息。
    packInfoList = []
    grePackets = []
    for packet in mirrorPackets:
        if packet.haslayer(GRE):
            # 判断 GRE 乘客协议是否是 ERSPAN
            if packet[GRE].proto == 0x88be:
                packInfo = {
                    "macSrc": "",
                    "macDst": "",
                    "ipSrc": "",
                    "ipDst": "",
                    "ipProto": "",
                    "portSrc": "",
                    "portDst": "",
                }
                erspanPack = packet[GRE][1]                    # 获取 GRE 内层的 ERSPAN 报文
                etherPack = erspanPack[1]                      # 获取 ERSPAN 报文内层的 Ethernet 报文
```

```
                macSrc = etherPack.src
                macDst = etherPack.dst
                if etherPack.type != 0x0800:
                    continue
                ipPack = etherPack[1]                    # 获取 Ethernet 报文上层的 IP 报文
                ipSrc = ipPack.src
                ipDst = ipPack.dst
                ipProto = ""
                # 如果是 icmp 报文
                if ipPack.proto == 1:
                    ipProto = "icmp"
                elif ipPack.proto == 6:
                    ipProto = "tcp"
                    packInfo["portDst"] = ipPack[1].dport
                    packInfo["portSrc"] = ipPack[1].sport
                else:
                    # 这边先只处理 icmp 报文和 tcp 报文，后面再增加 udp
                    continue
                packInfo["macSrc"] = macSrc
                packInfo["macDst"] = macDst
                packInfo["ipSrc"] = ipSrc
                packInfo["ipDst"] = ipDst
                packInfo["ipProto"] = ipProto
                if packInfo not in packInfoList:
                    packInfoList.append(packInfo)
    return packInfoList
```

步骤 6 main 方法

main 方法计算流量路径并输出，该部分代码先提示用户输入需要计算路径的流量五元组信息，再使用 Scapy 进行抓包。之后对网络中每一个开启远程镜像功能的设备进行实例化，并把所有 NetDevice 类实例添加到 devices 变量中。path_calc 方法以流量五元组信息，镜像的业务报文信息，开启远程镜像的设备为参数，计算出流量路径并输出。

```
def main():
    flow_info = input_flow_info()
    packInfoList = packets_capture()
    devices = []
    CE_B = NetDevice("CE-B", "10.154.186.102", "python", "Huawei12#$", interfaceConnect_CE_B)
    CE_D = NetDevice("CE-D", "10.154.186.104", "python", "Huawei@1234", interfaceConnect_CE_D)
    devices.append(CE_B)
    devices.append(CE_D)
    path_calc(flow_info, packInfoList, devices)

if __name__ == "__main__":
    main()
```

8.5 思考题

本实验是通过比较业务报文源目 MAC 地址与网络设备的 MAC 地址信息，来确定报文经过的网络设备及接口信息，但通过查找 MAC 地址表和 ARP 表的方式确定报文出入接口信息，存在错误的可能，因为脚本中是先抓包，待抓包结束后再获取网络设备的 MAC 地址表和 ARP 表信息，期间 MAC 地址表和 ARP 表可能会发生变化。请思考是否还有更简单准确的方法确定业务报文经过的网络设备及接口信息?

思考题参考答案

《SSH 实验》参考答案：

可以使用循环实现使用 SSH 登录多台设备查看配置。

《SNMP 自动化配置实验》参考答案：

1.自定义函数，结果略。

2.可以使用多线程，请参考《Python 编程基础》相应章节。

《NETCONF 配置实验》参考答案：

1.使用 if __name__ == '__main__'作用是当此代码被作为模块导入时，此代码块不被执行。

2.NETCONF 查询对象是 YANG，SNMP 查询对象是 MIB。

《配置文件对比实验》参考答案：

1.略。

2.登录设备部分代码可以单独封装成函数，被写入配置和获取配置功能调用。

3.引入时间模块实现定时功能；使用 NETCONF 更为高效获取设备配置文件。

《gRPC 远程查询配置实验》参考答案：

略。

《Telemetry 配置实验》参考答案：

1.完成数据过大情况下 gRPC 压缩实现更为高效的传输。

2.根据真实的网络应用需求，可以考虑选择同步或者异步方法实现数据收集和解析。前者为收集时立刻解析，后者为收集后统一解析。

《OPS 实验》参考答案：

设备 ZTP 流程为设备（无配置文件和插入 U 盘）从 DHCP 服务器处获取临时 IP 和文件服务器地址信息，然后从文件服务器获得配置脚本（boot.py），最后运行此脚本。脚本中实现设备初始化相关功能，例如获取版本、查看配置文件、配置 IP 地址等。

《网络流量分析实验》参考答案：

网络设备在配置三层远程镜像时，可以指定三层远程镜像的编号 erspan-id。命令如下
observe-port [observe-port-index] [interface interface-type interface-number]
destination-ip dest-ip-address source-ip source-ip-address [dscp dscp-value]
[erspan-id erspan-id]，erspan-id 参数被封装在镜像报文中，用于区分不同的镜像报文，取

值范围是 0~1023。可以利用 erspan-id 来确定被镜像的业务报文所经过的网络设备及接口信息。如用于本实验，CE-B 远程镜像功能可配置如下：

```
<CE-B>system-view immediately
Enter system view, return user view with return command.
[CE-B]observe-port 1 destination-ip 10.166.231.109 source-ip 10.154.186.102 erspan-id 1
[CE-B]observe-port 2 destination-ip 10.166.231.109 source-ip 10.154.186.102 erspan-id 2
[CE-B]observe-port 3 destination-ip 10.166.231.109 source-ip 10.154.186.102 erspan-id 3
[CE-B]observe-port 4 destination-ip 10.166.231.109 source-ip 10.154.186.102 erspan-id 4
[CE-B]interface 10GE 1/0/17
[CE-B-10GE1/0/17] port-mirroring observe-port 1 inbound
[CE-B-10GE1/0/17] port-mirroring observe-port 2 outbound
[CE-B-10GE1/0/17]interface 10GE 1/0/16
[CE-B-10GE1/0/16] port-mirroring observe-port 3 inbound
[CE-B-10GE1/0/16] port-mirroring observe-port 4 outbound
[CE-B-10GE1/0/16]quit
```

若分析发现远程镜像报文中的 erspan-id 为 2，即可确定业务报文是从 CE-B 的 10GE1/0/17 outbound 方向发送出去的。

CE-D 远程镜像功能配置，erspan-id 可以取值为 5~8，具体配置略。两台设备的 erspan-id 取值是不同的，所以通过 erspan-id 也直接确定了报文经过的网络设备。

华为认证系列教程

HCIP-Datacom-Network Automation Developer

NCE北向开放

实验指导手册

版本:1.0

华为技术有限公司

商标声明

 和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： http://e.huawei.com

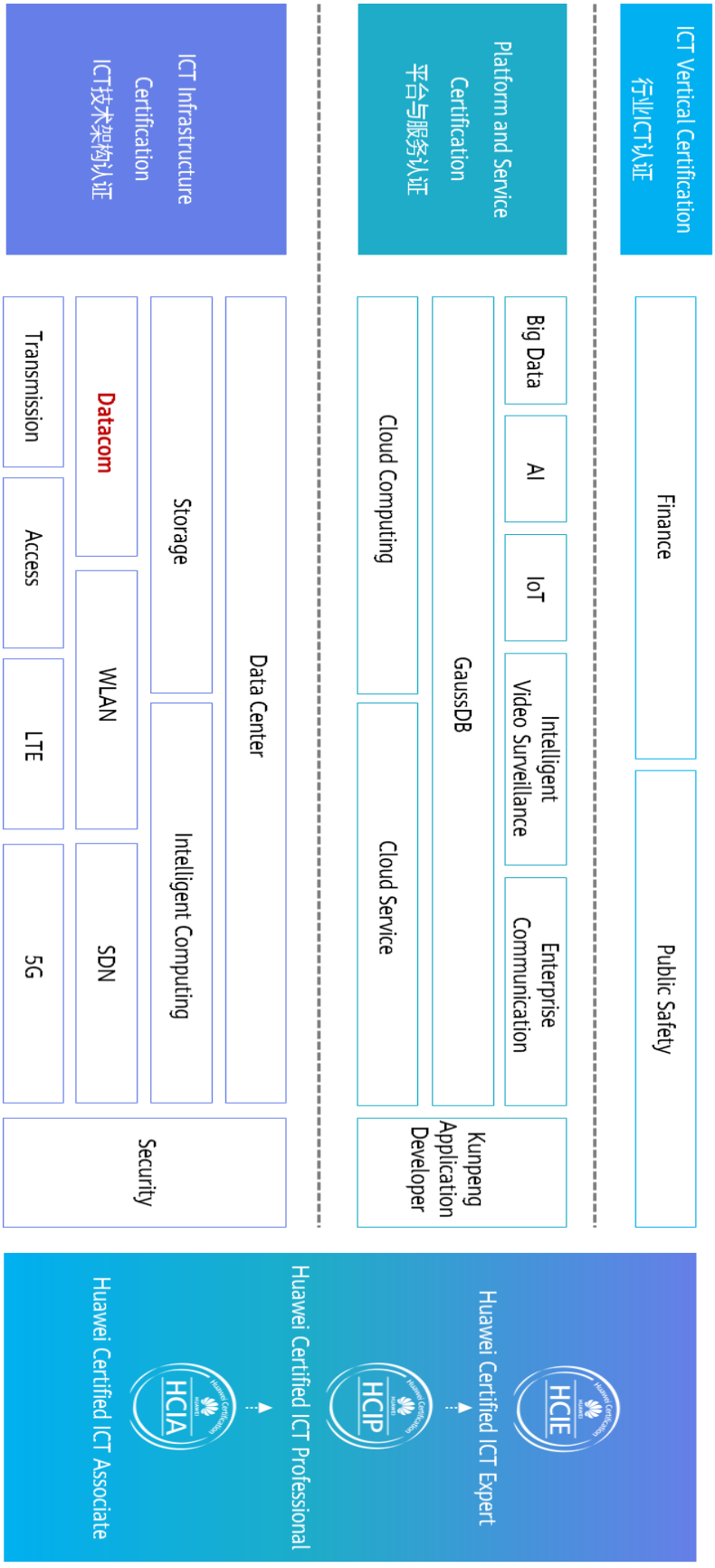
华为认证体系介绍

华为认证是华为公司基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，打造的ICT技术架构认证、平台与服务认证、行业ICT认证三类认证，是业界唯一覆盖ICT（Information and Communications Technology 信息通信技术）全技术领域的认证体系。

根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态。

HCIP-Datacom-Network Automation Developer定位于培养数通网络领域具备网络自动化开发专业知识和技能水平的高级工程师。通过HCIP-Datacom-Network Automation Developer认证将证明您能够胜任企业网络自动化开发工程师岗位，具备使用华为数通设备进行企业网络自动化部署、开发和运维的能力。

华为认证协助您打开行业之窗，开启改变之门，屹立在数通领域的潮头浪尖！


Huawei Certification
ICT Vertical Certification
行业ICT认证
Finance
Public Safety
Platform and Service Certification
平台与服务认证
Big Data
AI
IoT
Intelligent Video Surveillance
Enterprise Communication
GaussDB
Cloud Computing
Cloud Service
Kunpeng Application Developer
ICT Infrastructure Certification
ICT技术架构认证
Data Center
Storage
Intelligent Computing
Datacom
WLAN
SDN
Transmission
Access
LTE
5G
Security
HCIE
Huawei Certified ICT Expert
HCIP
Huawei Certified ICT Professional
HCIA
Huawei Certified ICT Associate

前 言

简介

本书为 HCIP-Datacom-Network Automation Developer 认证培训教程，适用于准备参加 HCIP-Datacom-Network Automation Developer 考试的学员，或者希望了解华为 iMaster NCE 北向开放基础知识和实践的读者。

读者知识背景

本文档主要适用于进阶学习的网络自动化工程师。读者需具备以下知识和技能：

- Python 编程基础
- 华为云代码托管
- RESTful 原理与实践
- 了解华为 iMaster NCE 相关产品

实验环境说明

环境介绍

本实验环境基于智简网络开发者社区。

智简网络开发者社区是面向数据通信领域依托华为公有云 DevCloud 开发服务，提供开发者和合作伙伴的“学习、开发、测试、交流”一站式服务平台。目前提供华为智简园区网络，智简数据中心网络，广域网络三大解决方案的开放场景，以及 API Explorer、API Studio、沙箱、DevOps 开发 IDE 与 SDK 等开发资源，帮助开发者快速体验、开发、集成和上线各行业应用。

本手册将介绍使用开发者社区环境进行智简园区网络和智简数据中心网络北向开放实践。

使用说明

- 注册华为云帐号并实名认证：https://www.huaweicloud.com/。

- 访问智简网络开发者社区：https://developer.huaweicloud.com/resource/network.html，获取更多信息。

目录

1 RESTful API 调用实践

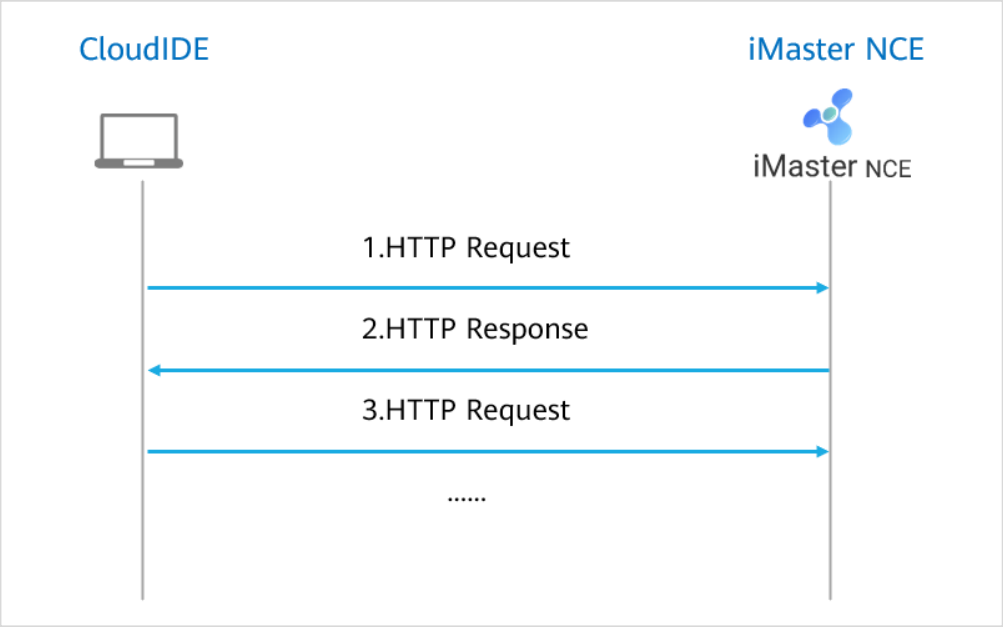
1.1 实验背景

iMaster NCE 是华为集管理、控制、分析和 AI 智能功能于一体的网络自动化与智能化平台。在 SDN 网络架构中，由控制器统一纳管网络设备。用户可以根据控制器北向开放接口实现意图驱动网络。

本实验通过代码实践，帮助读者快速掌握使用 Python requests 模块登录 iMaster NCE 并调用其业务接口。

1.2 实验介绍

1.2.1 组网说明



本实验组网包含两个对象，CloudIDE 和 iMaster NCE：

- CloudIDE 是由华为云提供的云端开发环境。读者也可以使用本地环境编写代码。
- iMaster NCE 可由数通开发者社区提供沙箱环境。

1.2.2 实验步骤

本实验步骤如下：

1. 环境准备：准备 IDE 和实验沙箱环境。
2. 编写 Python RESTful API 调用示例代码。
3. 结果验证。

1.3 环境准备

1.3.1 IDE 准备

本章节完整代码已上传到华为云 DevCloud。读者可以使用华为 CloudIDE 在线编程或将代码下载至本地编程：https://devcloud.cn-north-4.huaweicloud.com/codehub/project/68494a8ad06b4eea9a1c3f18be115161/codehub/620653/home。

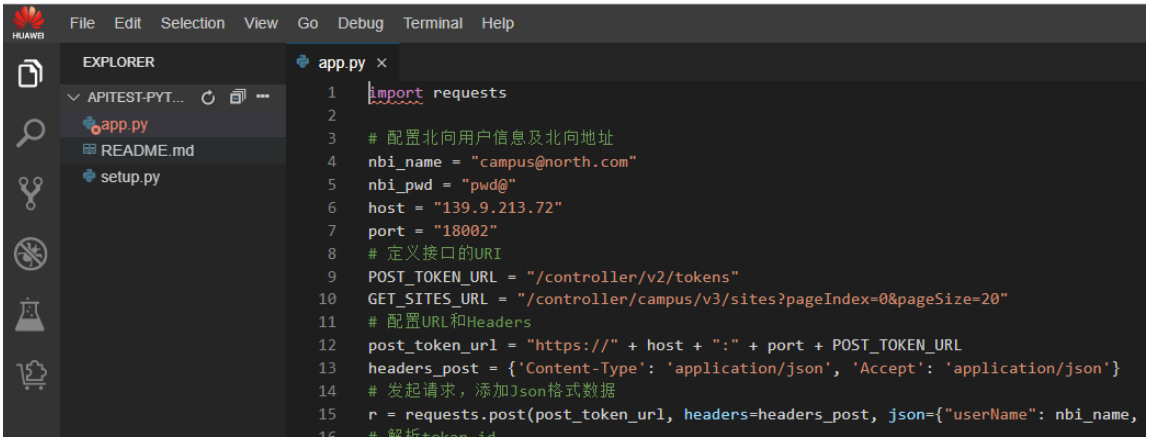
1.访问 CodeHub 控制台，新建项目和云上代码仓库。

更多操作请参考 CodeHub 指南，https://support.huaweicloud.com/codehub/index.html。

2.将完整代码 Fork 到仓库中。

3.点击 CloudIDE 即可打开代码。

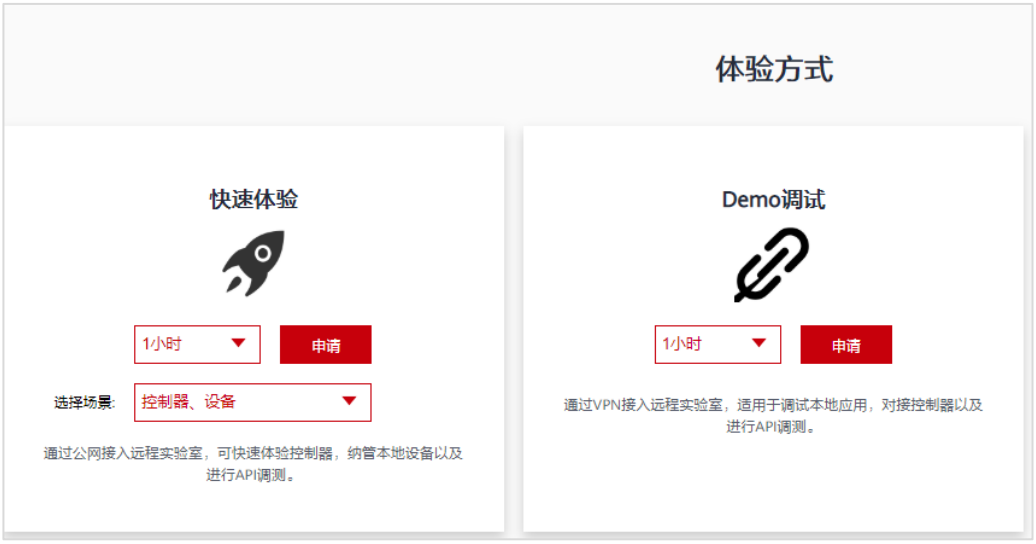
4.CloudIDE 界面显示如下：



1.3.2 沙箱预约

开发者社区提供华为数通解决的沙箱体验，本实验申请智简园区网络沙箱。

沙箱提供两种体验方式，快速体验和 Demo 调测：



- 快速体验使用公网接入实验室，适用于 CloudIDE 开发代码；
- Demo 调试使用 VPN 接入实验室，适用于本地 IDE 开发代码。

快速体验下选择“控制器、设备”和时长，点击申请，即可获得环境登录信息。

“租户账号”为用户 NCE 页面登录账户。

“北向账号”为位置定位应用与 NCE 交互的账号。用于组网说明中 1-3 交互步骤。

“北向 URL”为 iMaster NCE 登录地址，点击“前往体验”可跳转。

“密码”为本次申请的沙箱环境的登录密码。

至此，环境准备完成。

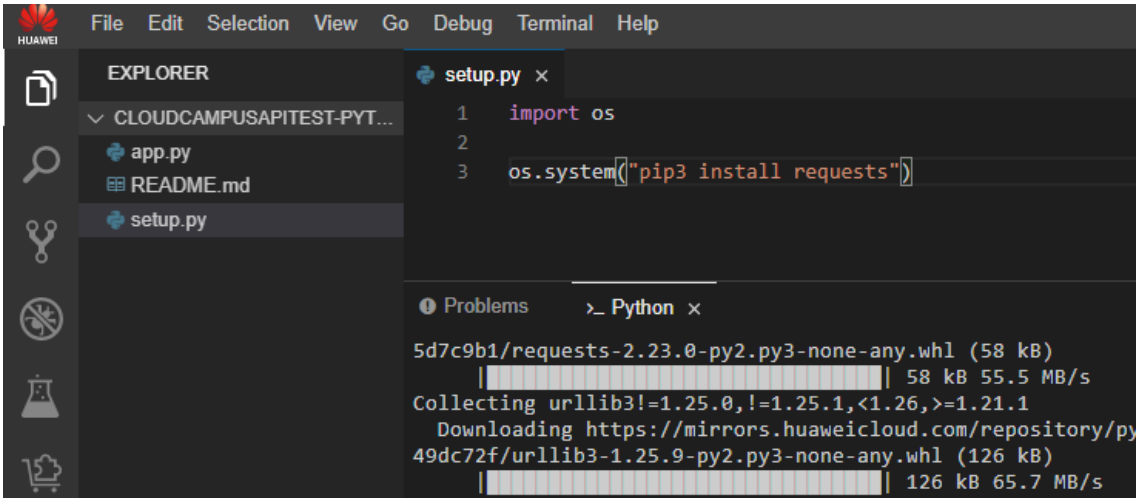
1.4 编写 Python 代码

本例中 Python 代码与 iMaster NCE 交互过程如图。首先编写代码发起 POST 请求获取认证信息 Token，发起 GET 请求（携带 Token）获取站点信息。

1.4.1 安装 requests 模块

本实验需要使用 Python requests 模块。运行 setup.py 代码安装 requests。

注：可点击鼠标右键选择“Run Python File in Terminal”运行代码。



1.4.2 配置对接信息

配置 iMaster NCE 信息。

填入“沙箱预约”参数，包括北向用户名、密码、IP 地址和端口。

```
import requests

# 配置北向用户信息及北向地址
nbi_name = "campus02@north.com"
nbi_pwd = "5aAZ4lh44@"
host = "139.9.213.72"
port = "18002"
```

1.4.3 获取登录认证

iMaster NCE 通过租户账户登录来确认使用者的权限。客户端（Python 代码）向服务器发送登录认证请求，服务器会返回一个 Token 和过期时间。

本章节需参考指定版本的 iMaster NCE-Campus 产品文档“RESTful API 开发指南”可以获得认证登录的 URI 和请求头和 Body 相关信息。例如：

1.发起 POST 请求。

本例中 URI 为“/controller/v2/tokens”，请求头携带“Content-Type”和“Accept”，Body 携带“userName”和“password”字段。

```
# 定义接口的 URI
POST_TOKEN_URL = "/controller/v2/tokens"

# 配置 URL 和 Headers
post_token_url = "https://" + host + ":" + port + POST_TOKEN_URL
headers_post = {'Content-Type': 'application/json', 'Accept': 'application/json'}
# 发起请求，添加 Json 格式数据
r = requests.post(post_token_url, headers=headers_post, json={"userName": nbi_name, "password": nbi_pwd}, verify=False)
```

根据开发指南要求 HTTP 操作为 POST。代码中使用 requests.post 方法发起请求。

更多 requests 用法请参考文档，https://requests.readthedocs.io/en/master/。

2.获取并打印 Token。

解析 iMaster NCE 返回的 Response。

```
# 解析 token_id
token_id = r.json()['data']['token_id']
print("1.【Get Token Id】")
print("【post_token_url】: "+post_token_url)
print("【token_id】: "+token_id)
```

输出结果为：

```
1.【Get Token Id】
【post_token_url】：https://139.9.213.72:18002/controller/v2/tokens
【token_id】：x-
rs5jvy7z47vsum042r2qc9vtaq44o9hgqohf3uca7z9clio5vvvz86al3s5dvs0aphpck5defw45uqpi3xdgddmn482kgajutf7vlf
s649rw6k5c6r6l0a5i1g4b5j9c
```

1.4.4 请求 iMaster NCE 站点信息

使用 GET 请求调用 iMaster NCE-Campus 接口获取站点信息。

1.获取站点信息 URI。

查询 API 手册对应 URI。

```
# 定义接口的 URI

GET_SITES_URL = "/controller/campus/v3/sites"
```

2.设置 GET 请求属性。

设置 URL 和 Header 等信息，注意要将 token_id 放在 Header 中。

```
# 配置 URL 和 Headers
get_sites_url = "https://" + host + ":" + port + GET_SITES_URL

headers_get = {'Content-Type': 'application/json', 'Accept': 'application/json', 'X-AUTH-TOKEN':token_id}
```

X-AUTH-TOKEN 字段携带 token_id。

3.发起 GET 请求。

```
# 发起请求
r = requests.get(get_sites_url, headers=headers_get, verify=False)
```

4.解析响应并输出结果。

```
# 解析站点信息
print("2.【Get Sites Info】")
```

```
print("【get_sites_url】: "+get_sites_url)
total_records = r.json()['totalRecords']
print("【total_records】: "+str(total_records))
print(r.text)
```

输出结果为：

```
2.【Get Sites Info】
【get_sites_url】 : https://139.9.213.72:18002/controller/campus/v3/sites
【total_records】 : 1
{"errcode":"0","errmsg":"","totalRecords":1,"pageIndex":1,"pageSize":20,"data":[{"id":"95e61c0c-ed50-4683-9fb3-30dcbfca3c69","tenantId":"687fc863-8aaa-4c28-ba3a-e912a6b52e77","name":"site01","description":"","type":["AP"]}]}
```

可知 iMaster NCE-Campus 上有站点“stie01”类型为“AP”。

1.5 结果验证

登录 iMaster NCE-Campus 验证信息。

点击“前往体验”。

输入租户账号密码，完成登录。

点击“设计”>“站点管理”，查看当前站点信息。

至此本实验示例结束。

后续开发者可以根据业务需求调用 iMaster NCE 相关 API 实现业务发放。

1.6 思考题

本例中 Python 代码和 iMaster NCE-Fabric 有几次交互？分别有什么不同？

2 智简园区无线定位实践

2.1 实验背景

华为智简园区网络（CloudCampus）解决方案，应用前沿的有线和无线技术，加持大数据、AI和云技术，以业务为中心，构建万物互联、业务无忧和可平滑演进的园区网络，使能行业数字化转型。

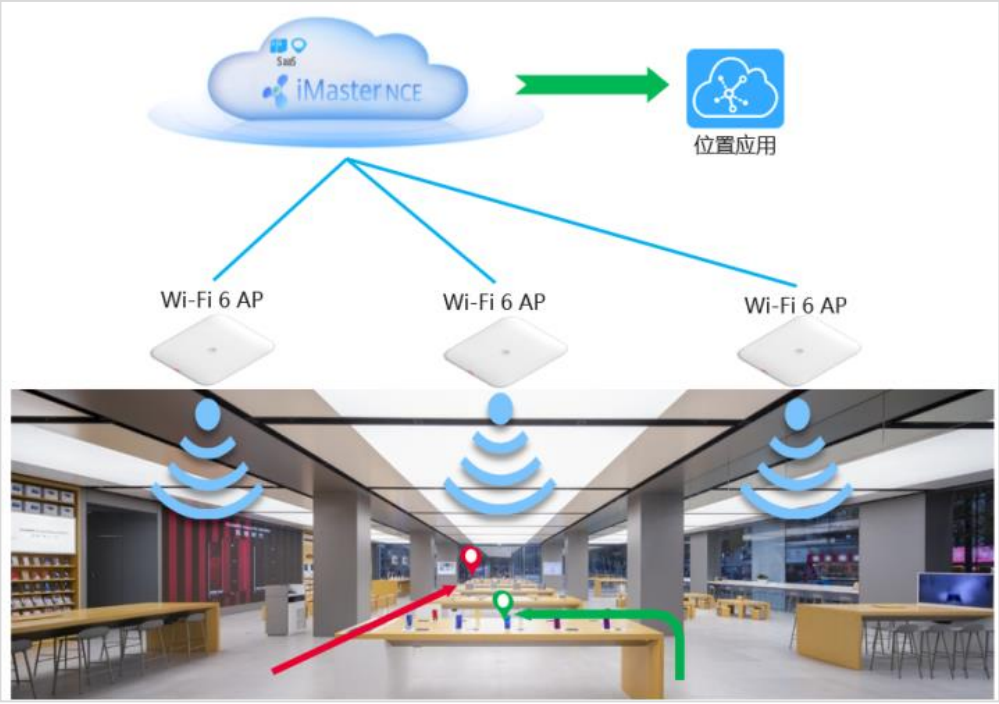
本实验将指导读者开发位置服务应用（Location Based Service，LBS），调用华为 iMaster NCE-Campus 的位置服务 API 和增值业务 API，统计 Wi-Fi 用户位置信息。本课程您会学习：

- 对接 iMaster NCE-Campus 的位置服务接口
- 调用 iMaster NCE-Campus 的增值业务接口
- 解析 iMaster NCE-Campus 的位置数据

2.1.1 位置服务介绍

位置服务（LBS，Location Based Services）是利用各类型的定位技术来获取定位设备当前的所在位置，通过移动互联网向定位设备提供信息资源和基础服务。

华为 CloudCampus 位置服务（Wi-Fi 方案），获取基于 Wi-Fi 的位置数据，并上报至开发者的 LBS 应用。开发者可以使用数据，计算范围内的 Wi-Fi 终端（无论关联与否）位置信息。

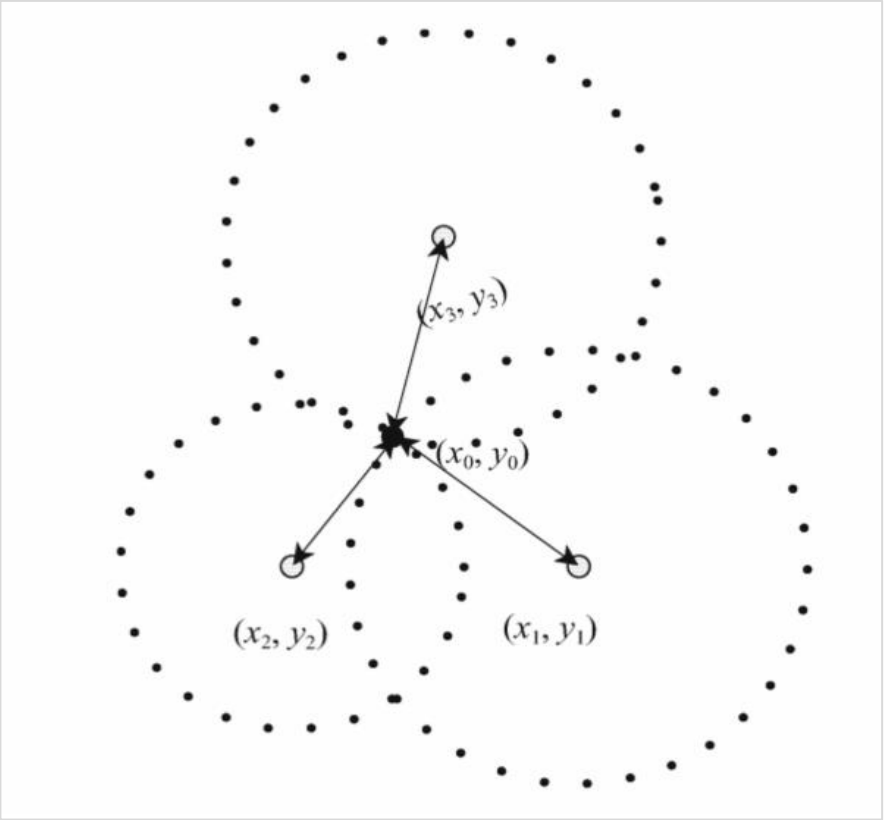


[位置服务]与[增值业务]结合可以实现的功能有：实时定位，室内导航，客流分析，用户画像，资产监控与管理，营销推送，位置热图，轨迹跟踪等增值应用。

2.1.2 Wi-Fi 定位介绍

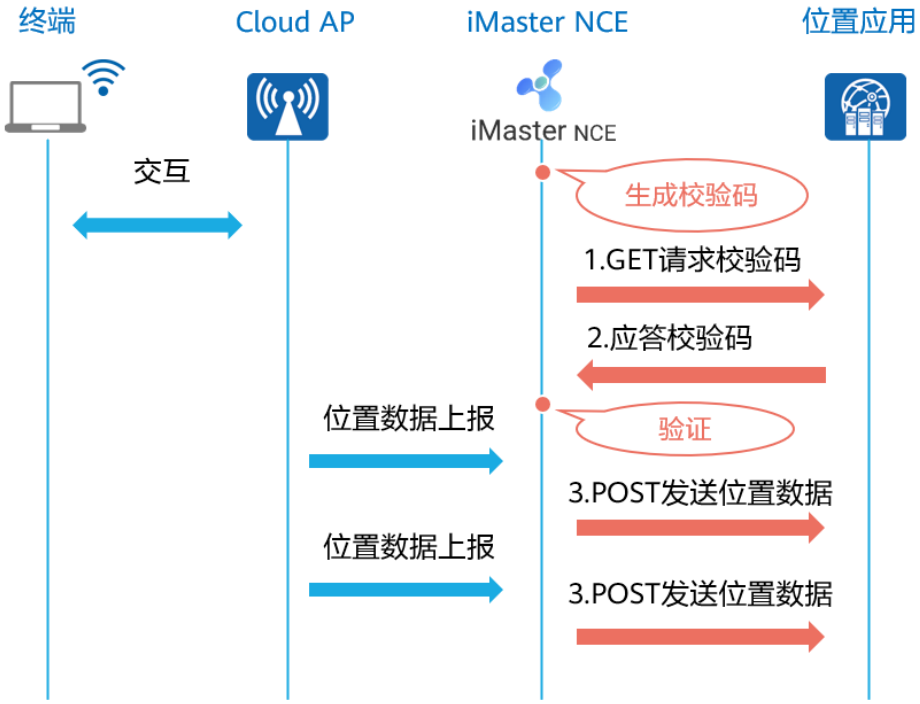
通过 Wi-Fi 提供位置服务是比较流行的一种室内定位技术，其定位方法是基于 RSSI 信号衰弱模型的三边测量定位法。

首先根据 RSSI 信号强度计算出终端距离 AP 的距离。在同时有超过 3 个 AP 时，以 AP 为圆心交点为终端位置。此场景可以抽象成在已知超过三个定点的位置(x1,y1),(x2,y2),(x3,y3)…和未知点(x0,y0)到三点的距离 l1,l2,l3…，求未知点(x0,y0)的坐标的算法模型。


(x_3, y_3)
(x_0, y_0)
(x_2, y_2)
(x_1, y_1)

2.2 实验介绍

2.2.1 组网说明



本实验组网包含四个对象，终端、CloudAP、iMaster NCE 和位置应用。其中终端、CloudAP、iMaster NCE 可由开发者社区提供沙箱环境，读者需编写位置应用代码。位置应用代码推荐使用华为云 CloudIDE 运行。

2.2.2 实验步骤

本实验步骤如下：

1. 环境准备：准备 IDE 和实验沙箱环境。
2. 位置定位应用开发：编写位置定位应用代码。
3. 结果验证：应用对接 iMaster NCE，验证结果。

2.3 环境准备

2.3.1 IDE 准备

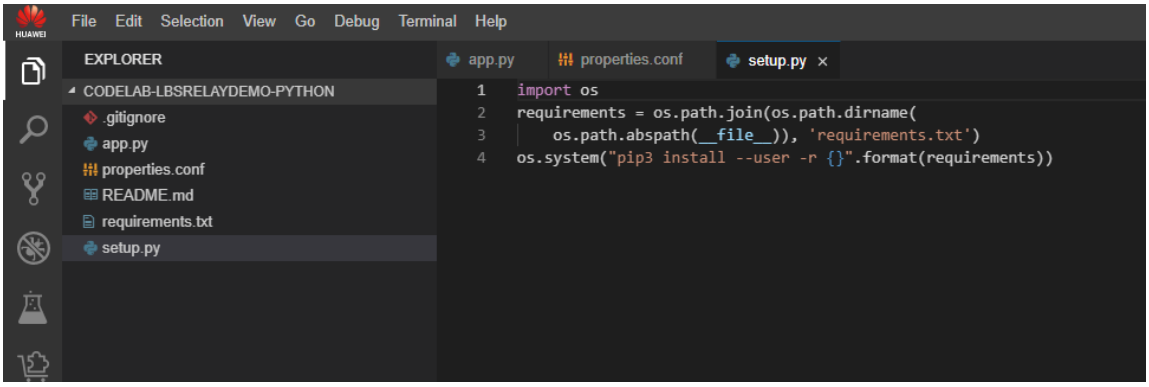
本章节完整代码已上传到华为云 DevCloud。读者可以使用华为 CloudIDE 在线编程或将代码下载至本地编程：https://devcloud.cn-north-4.huaweicloud.com/codehub/project/68494a8ad06b4eea9a1c3f18be115161/codehub/577707/home。

1.访问 CodeHub 控制台，新建项目和云上代码仓库。

更多操作请参考 CodeHub 指南，https://support.huaweicloud.com/codehub/index.html。

2.将完整代码 Fork 到仓库中，点击 CloudIDE 即可打开代码。

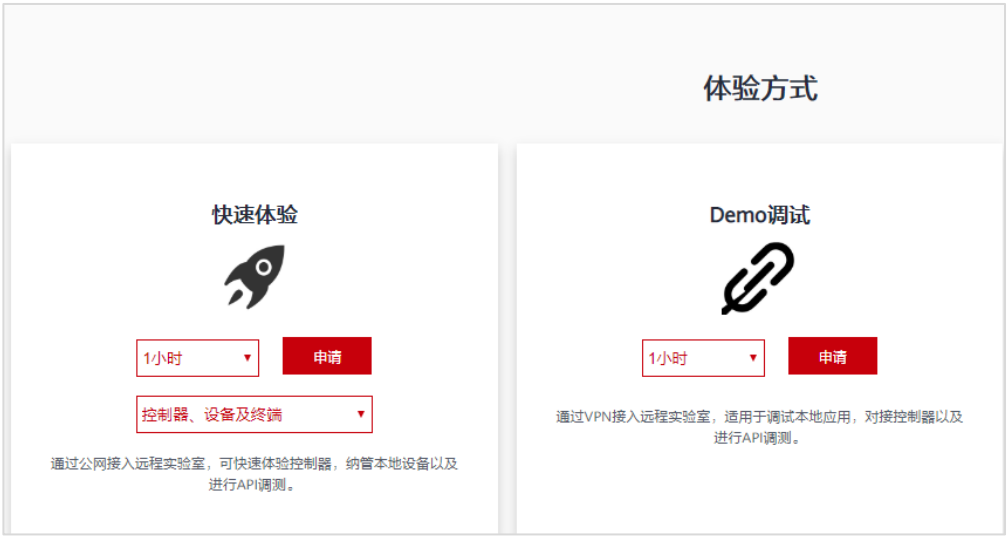
3.CloudIDE 界面显示如下：



2.3.2 沙箱预约

开发者社区提供华为数通解决的沙箱体验，本实验申请智简园区网络沙箱。

沙箱提供两种体验方式，快速体验和 Demo 调测：



- 快速体验使用公网接入实验室，适用于 CloudIDE 开发代码；
- Demo 调试使用 VPN 接入实验室，适用于本地 IDE 开发代码。

快速体验下选择“控制器、设备及终端”和时长，点击申请，即可获得环境登录信息。

“租户账号”为用户 NCE 页面登录账户。

“北向账号”为位置定位应用与 NCE 交互的账号。组网说明中 1-3 步骤。

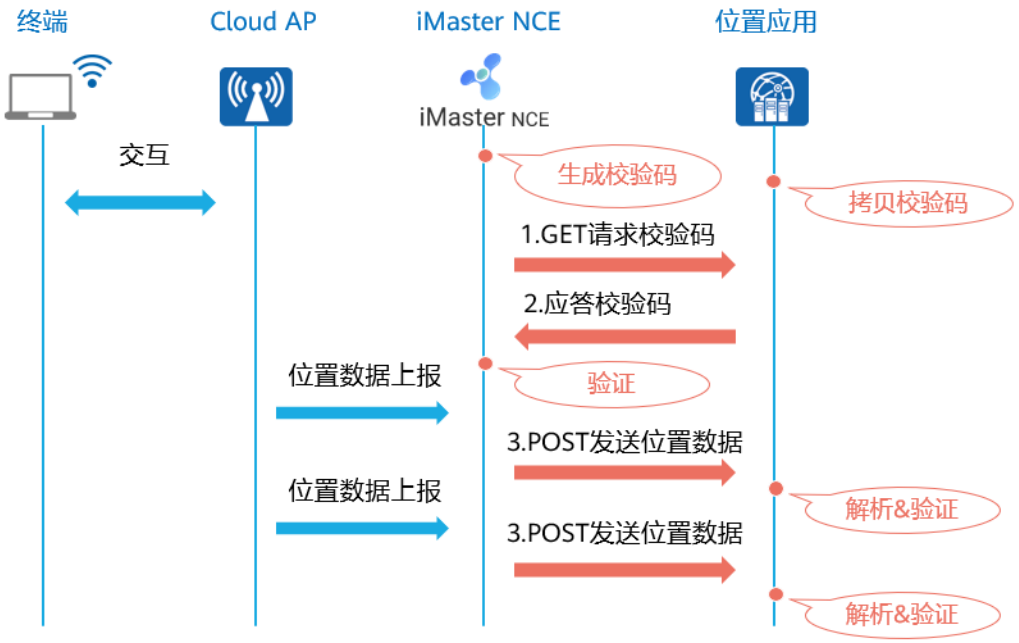
“北向 URL”为 iMaster NCE 登录地址，点击“前往体验”可跳转。

“密码”为本次申请的沙箱环境的登录密码。

至此，环境准备完成。

2.4 位置定位应用开发

本章节您将尝试编写一个 LBS 应用 Demo，学会如何调用 iMaster NCE-Campus 位置服务接口实现终端位置信息数据上送，并通过调用 iMaster NCE-Campus 增值业务 API 实现查询用户信息和流量统计。



对于位置应用和 iMaster NCE 的交互流程如图：

- NCE 生成校验码，拷贝此验证码到位置应用。
- 由 NCE 发起 1&2 请求和应答完成校验。
- 通过验证后，NCE 发送位置数据。
- 位置应用解析并验证数据。

位置应用开发步骤有：

1. 安装 iMaster NCE-Campus Python SDK。
2. 编写应答校验交互代码。
3. 编写解析终端数据代码。
4. 编写用户查询和流量统计代码。

2.4.1 安装 iMaster NCE-Campus SDK

代码仓库中 requirements.txt 已写入依赖包，完整信息如下：

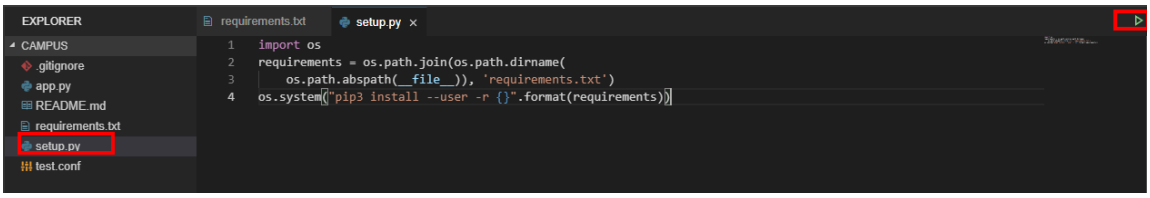
```
certifi >= 14.05.14
six >= 1.10
python_dateutil >= 2.5.3
setuptools >= 21.0.0
urllib3 >= 1.15.1
flask >= 1.1.1
requests >=2.22.0

-e git+https://codehub.devcloud.cn-north-4.huaweicloud.com/campus00001/CloudCampusPythonSDK.git@master#egg=cloudcampus
```

更多智简园区资源信息请参考 https://devzone.huawei.com/apistudio/sample/campus/apiSdk.html。

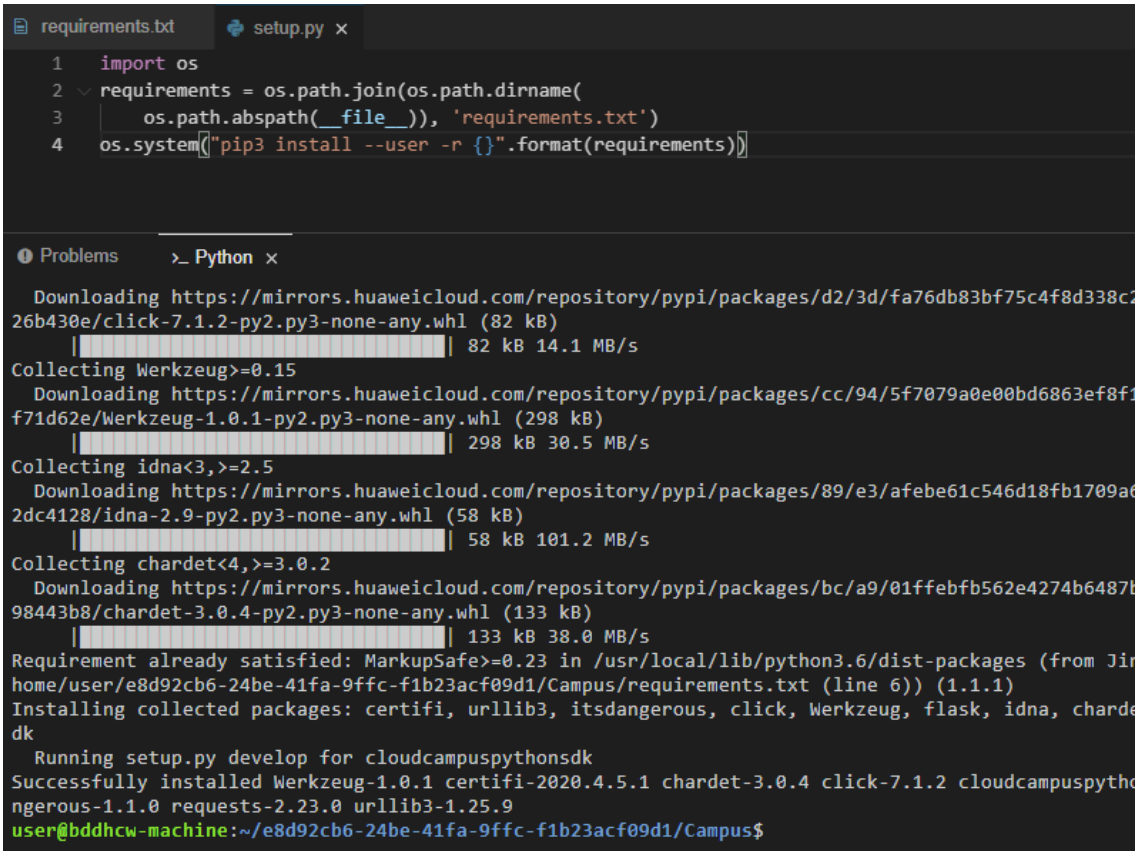
setup.py 包含安装代码。

```
import os
requirements = os.path.join(os.path.dirname(
      os.path.abspath(__file__)), 'requirements.txt')
os.system("pip3 install --user -r {}".format(requirements))
```

运行 setup.py 代码安装 SDK（CloudIDE 在 setup.py 中右键单击“Run Python File in Terminal”或点击运行即可）。



安装成功。



本地获取 SDK 请参考“智简园区网络”>“资源下载”。

2.4.2 应答校验交互代码

iMaster NCE-Campus 发起请求，LBS 应用需要进行正确的应答。对应流程图中流程 1，2。

1.在 properties 文件（properties.conf）中创建变量：

```
[LBS]
validator =
secret = Codelab
```

validator 值将由 iMaster NCE-Campus 生成校验码拷贝至此。

secret 值为对接参数密码，用于和 NCE 对接。本例为“Codelab”

2.创建 web 应用。本应用使用 Flask 框架（app.py）：

```
app = Flask(__name__)

@app.route('/', methods=['GET', 'POST'])
def main():
    #
    #

if __name__ == "__main__":
    # Flask APP  启动
    app.run(
        host = "0.0.0.0",
        port = 8899
    )
```

3.创建 ValidatorData 类（app.py）：

```
class ValidatorData:
    #  存放 iMaster NCE-Campus 生成的 validator
    validator = ""

    def __init__(self, validator):
        self.validator = validator
```

4.响应来自 iMaster NCE-Campus 的请求（app.py）：

```
@app.route('/', methods=['GET', 'POST'])
def main():
    #读取 properties.conf 文件
    parser = configparser.ConfigParser()
    parser.read("properties.conf")
    if request.method == 'GET':
        #返回文件中存放的 validator 完成对接
        return json.dumps(ValidatorData(parser.get("LBS", "validator")).__dict__)
```

更多 Flask 请参考 https://flask.palletsprojects.com/en/1.1.x/。

2.4.3 解析终端数据代码

iMaster NCE-Campus 将 Wi-Fi 终端信息通过 POST 发出时，LBS 应用应解析数据，并存储与输出（对应图中流程 3）。

1.Wi-Fi 终端数据格式

请求体中的终端位置数据采用如下 JSON 格式：

```
{
    "data": [
      {
        "apMac": "4C:FA:CA:D8:23:A0",
        "terminalList": [
         {
             "terminalMac": "88:19:08:F1:88:45",
             "rssi": -68,
             "timestamp": 1557460789000
          },
         {
            "terminalMac": "90:2E:1C:6A:2A:57",
            "rssi": -57,
            "timestamp": 1557460789000
          }
        ]
      }
    ],
    "secret": "Codelab",
    "type": "ApLocation"
 }
```

Json字段	说明
apMac	终端接入点AP设备mac地址。
terminalMac	终端mac地址。
rssi	终端接收AP设备Wi-Fi信号场强值，单位dBm。
timestamp	接入点AP探测到终端信息的时间戳。
secret	凭据字段，由开发者提供，用作数据校验。
Type	上报数据类型，当前仅支持ApLocation。

关于对接原理详细介绍，请参见智简网络开发者社区>学习教程：(https://devzone.huawei.com/cn/enterprise/campus/apiDoc.html)。

2.创建.csv 文件存储（app.py）：

```
# 获取 Post Request
body = request.get_json()
print(body)# dict 类型
# 创建当前时间的 csv 文件
path = str(time.strftime("%Y%m%d-%H%M%S", time.localtime())) + ".csv"
with open(path, 'a', newline='') as f:
    csv_write = csv.writer(f)
    csv_head = ["terminalMac", "rssi", "timestamp", "apMac", "secret", "type"]
    csv_write.writerow(csv_head)
```

3.解析 iMaster NCE-Campus 上送的数据并保存至 csv 文件（app.py）：

```
path = str(time.strftime("%Y%m%d-%H%M%S", time.localtime())) + ".csv"
with open(path, 'a', newline='') as f:
    csv_write = csv.writer(f)
    # 生成各列列名
    csv_head = ["terminalMac", "rssi", "timestamp", "apMac", "secret", "type"]
    csv_write.writerow(csv_head)
    # 校对 secret 字段
    secret = body['secret']
    if secret != parser.get("LBS", "secret"):
        return ""

    type = body['type']
    # print(type)

    data_list = body['data']
    # print(data_list)

    # 循环解析 json 数据
    for data in data_list:
        ap_mac = data['apMac']
        # print(ap_mac)
        terminal_list = data['terminalList']
        # print(terminal_list)
        for terminal in terminal_list:
            rssi = terminal['rssi']
            # print(rssi)
            terminal_mac = terminal['terminalMac']
            # print(terminal_mac)
            timestamp = terminal['timestamp']
            # print(timestamp)
            csv_write.writerow([terminal_mac, rssi, timestamp, ap_mac, secret, type])
```

2.4.4 查询用户和流量统计代码

iMaster NCE-Campus 提供灵活的增值业务 API，可以作用户画像和分析，提供用户个性化的 Wi-Fi 服务。访问智简网络开发者社区增值业务场景，获取更多信息。

CloudCampus SDK 开发指南：
http://devzone.huawei.com/apistudio/sample/index.html?id=141&categoryType=campus#Python SDK 向导。

1.新建 API 客户端（app.py）

```
# init api
tenantName = 'demo15@north.com'
tenantPwd = 'uK7T1Ls4q@'
host = '139.9.213.72'
port = '18002'
config = Configuration(host, port, tenantName, tenantPwd)
api_client = ApiClient(config)
```

请注意，此处 tenantName 填写“沙箱预约”的“北向账号”，其他参数也按照实际预约情况填写。

2.查询站点信息（app.py）

站点是网络设备的管理集合，站点内的设备可以统一管理和运维。

站点名称根据沙箱环境的实际信息填写。使用租户账号登录 NCE 页面，查看站点信息。本例为 site01。

```
# 根据 Site name 查询 siteId
site_api = SiteManagerApi(api_client)
page_index = 1
page_size = 20
name = 'site01'
site_id = site_api.query_sites(page_index=page_index, page_size=page_size, name=name).data[0].id
print('siteId:    ' + site_id)
```

3.输出站点下的用户流量（app.py）

```
# 查询用户流量统计 并将用户和流量信息打印出来
station_api = StationOpenApiApi(api_client)
try:
    model =    station_api.query_site_station_info(site_id,1,20)
except ValueError as e :
    print('EXCEPT :            ' + str(e))
else:
    print('QUERY STATION INFO : SUCCESS')
    print('response body :            ' + str(model))
finally:
    print('QUERY STATION INFO END')
```

2.5 结果验证

结果验证内容为 LBS 应用与 iMaster NCE-Campus 位置服务接口的对接，解析并存储 WI-Fi 终端数据，以及统计 Wi-FI 用户的流量信息。

2.5.1 配置用户接入

2.5.1.1 配置 SSID

1.使用租户账户登录 iMaster-NCE。

登录地址见沙箱预约部分。

2.在主菜单选择“配置>站点配置>AP”。

3.选择“创建”，输入 SSID 名称，选择网络连接方式为“NAT”，单击“下一步”。

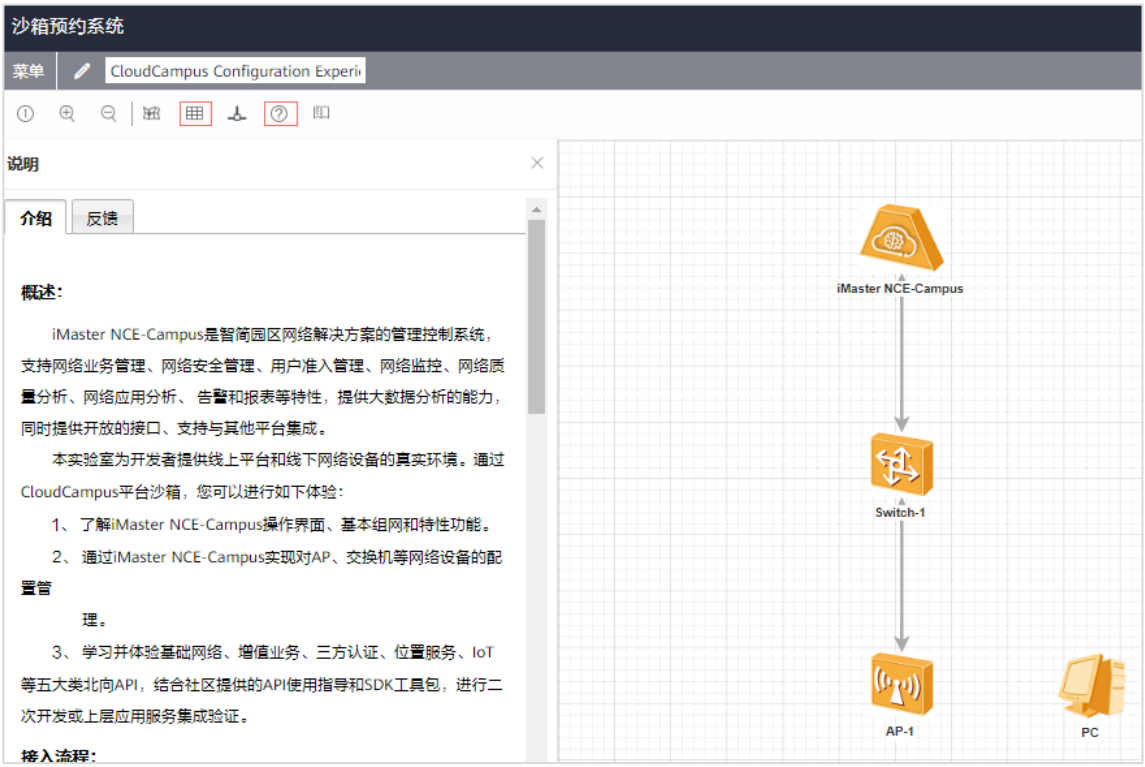
4.使用默认的安全认证配置，单击“下一步”；使用默认的策略控制配置，单击“确定”。

至此，SSID 配置结束。终端可以连接此无线网络，访问互联网。

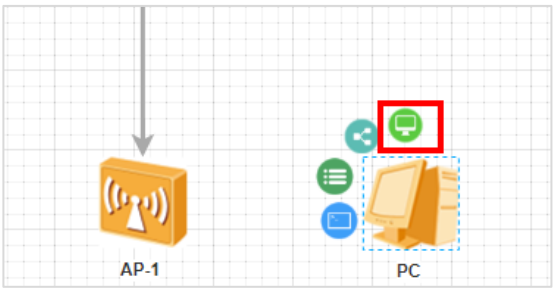
2.5.1.2 配置终端接入

在沙箱预约成功界面点击“进入拓扑”。

页面将自动跳转到拓扑界面。



选定终端连接。



可进入终端界面连接无线网络，然后访问互联网。

2.5.2 配置 iMaster NCE 对接应用

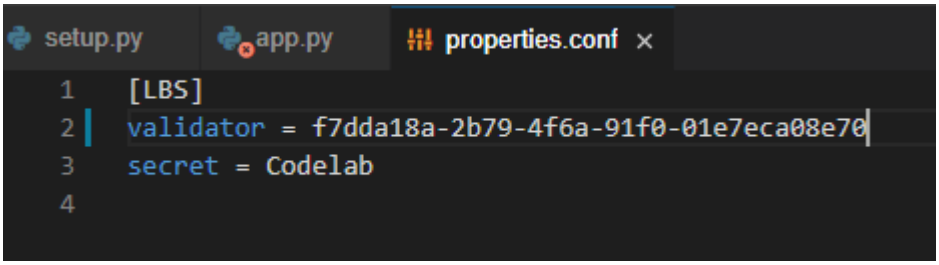
2.5.2.1 配置对接与启动应用

在 iMaster NCE-Campus“系统 > 系统设置 > 第三方服务 > 数据上传”页面配置数据对接。

1.数据类型选择“AP Location Data”。

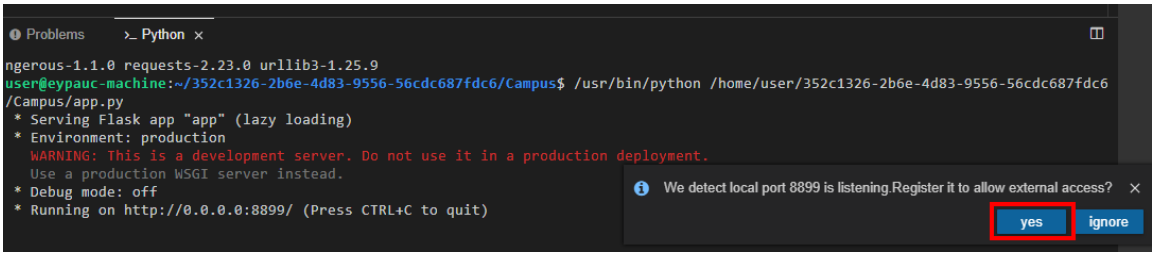
2.单击“校验码”右边“生成”按钮。文本框生成 UUID 格式的校验码。

3.复制校验码至 CloudIDE 的 properties.conf 文件。



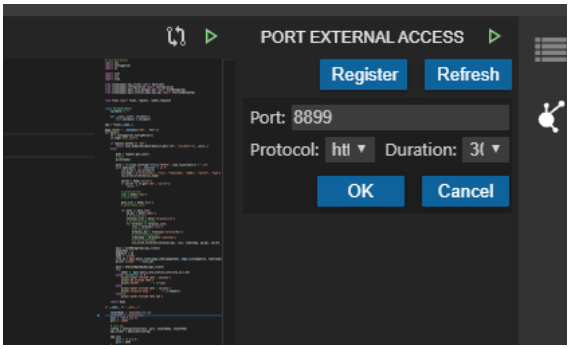
4.启动应用。

运行 CloudIDE 中 app.py。

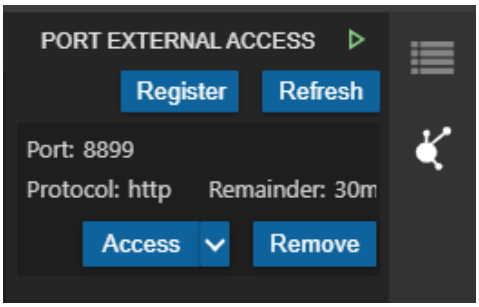


运行可见 Flask 应用已正常运行。为登录应用界面需要将程序映射至公网，CloudIDE 会弹出选择框。

点击“yes”后在 CloudIDE 右上角出现映射信息。



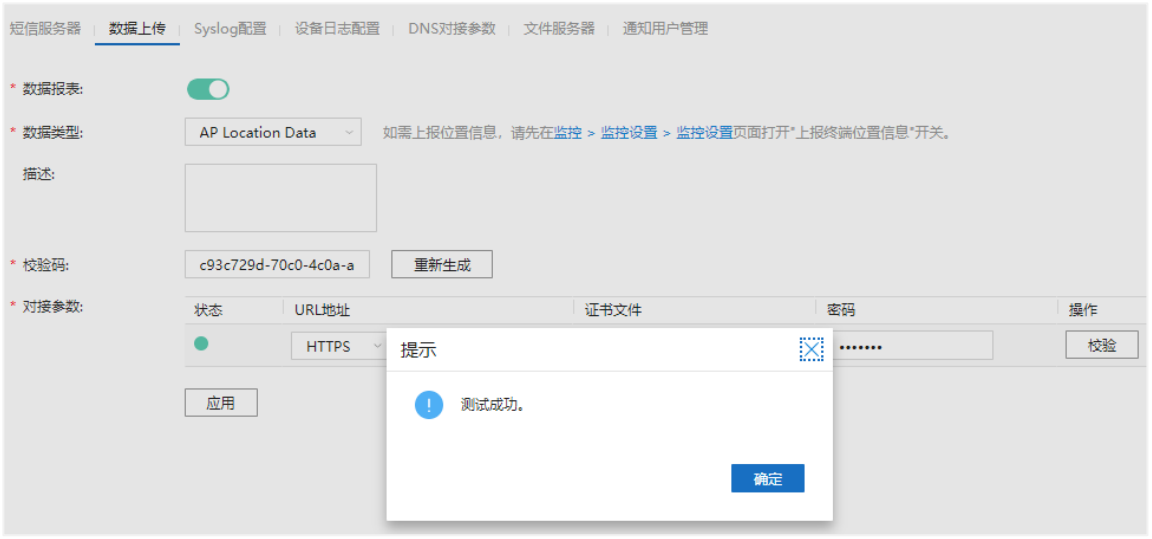
点击“OK”将变为“Access”。



点击“Access”，进入搭建的网站。

5.将程序对应的 url 复制到 iMaster NCE-Campus 界面，输入密码“Codelab”，单击“校验”。

若出现测试成功，则校验通过。

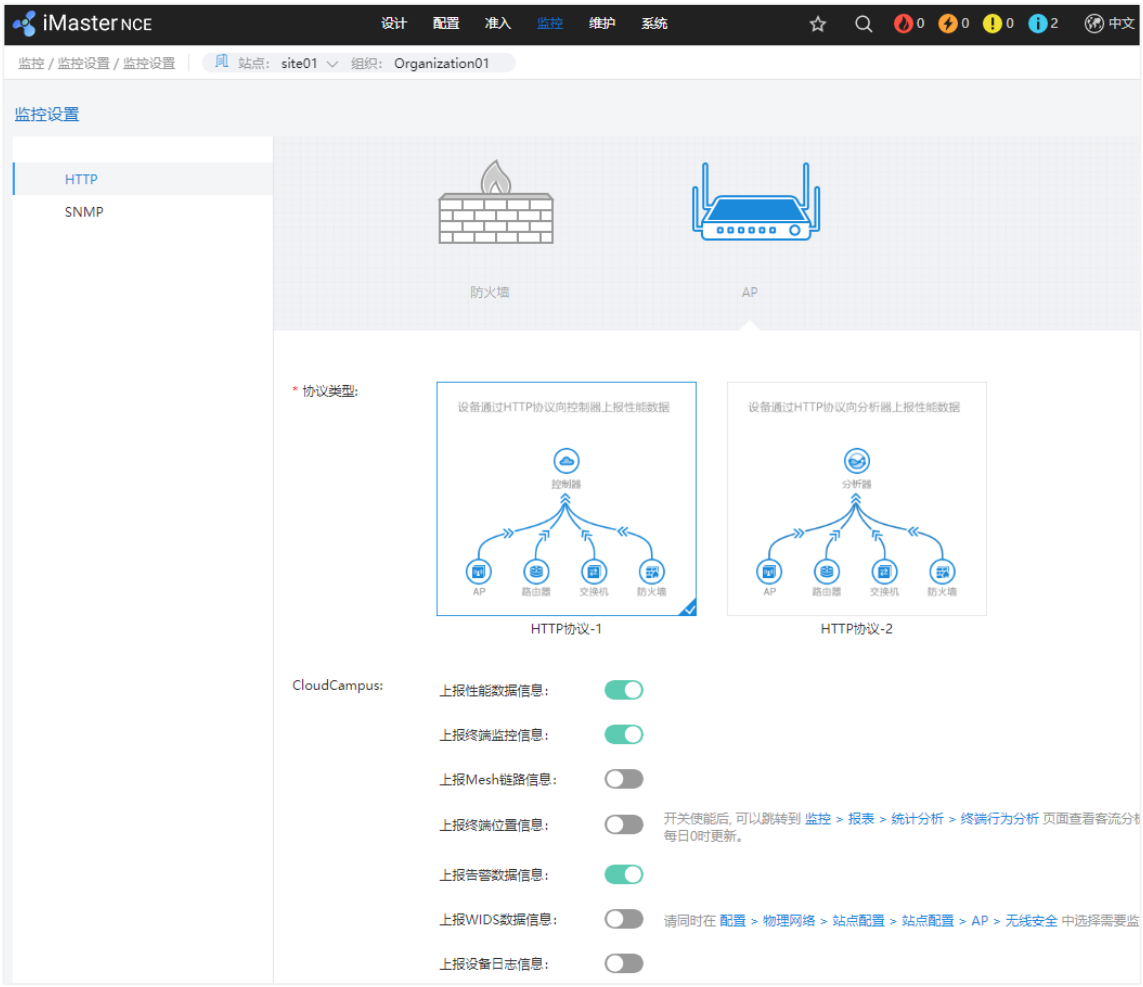


注意不要忘记点击“应用”。

2.5.2.2 使能数据上传服务

配置 NCE 对接位置应用前，需开启数据上传服务。

1.进入 iMaster NCE-Campus“监控>监控设置>HTTP>AP”页面。



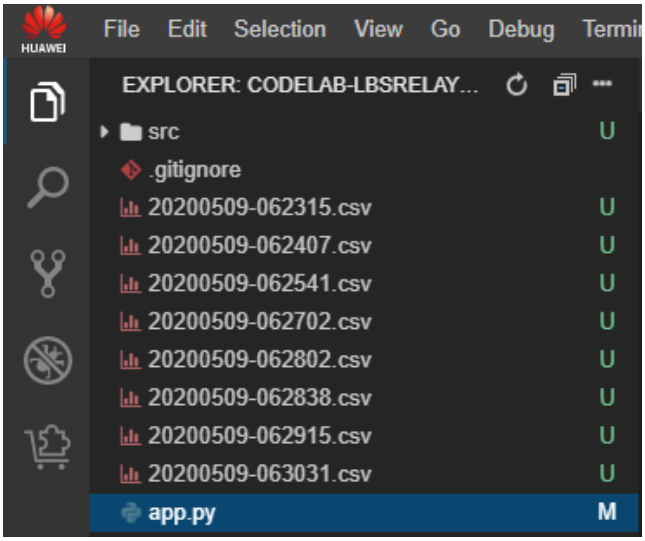
2.开启“上报终端位置信息”，点击保存。

2.5.3 查看数据

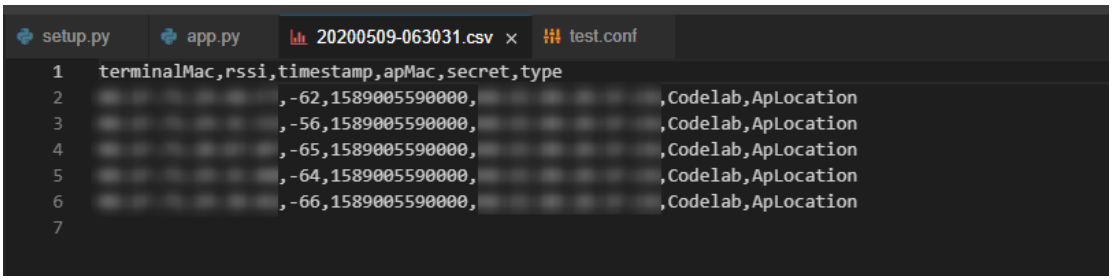
通过校验后，iMaster NCE-Campus 会将站点下 Cloud AP 扫描到的 Wi-Fi 终端信息上报至位置应用。

1.终端数据解析。

位置应用解析，在控制台打印并将数据存储为 csv 格式。



CSV 文件内容如下：



2.用户信息查看和流量统计。

```
x-84fwvx5hbtli7tmm06c949pf084bep4ahhdj9i7zbxjtmlulrvgb7w1gs5fzbzc6rt7z08g4l
QUERY STATION INFO : SUCCESS
response body :       {'data': [{'access_time': 1588822870,
           'access_type': 1,
           'account': 'e4****91',
           'auth_type': 'NOAUTH',
           'channel': 1,
           'cumulative_traffic': 0,
           'device_name': 'AP7050DE_18',
           'downward_speed': 3,
           'dual_frequency': 1,
           'frequency_band': 1,
           'host_name': 'HUAWEI_nova_2s-d2826d203c',
           'mode': 4,
           'online_status': 2,
           'online_time': 294,
           'package_loss_rate': 75,
           'port_index': 0,
           'retrans_rate': 37,
           'rssi': -91,
           'send_package_speed': 1000000,
           'signal_noise_ratio': 4,
           'ssid': 'CloudCampus-LBS',
           'sticky_tags': 1,
           'terminal_ip': '10****48',
           'terminal_mac': 'E4****91',
           'upward_speed': 4,
           'vlan': 3911},
          {'access_time': 1588839896,
           'access_type': 1,
           'account': '08****01',
           'auth_type': 'NOAUTH',
           'channel': 1,
           'cumulative_traffic': 0,
           'device_name': 'AP7050DE_18',
           'downward_speed': 52,
           'dual_frequency': 1,
           'frequency_band': 1,
```

至此，本无线定位应用实践结束。

2.6 思考题

获取到无线定位数据后如何处理？

3 智简园区第三方认证实践

3.1 实验背景

华为智简园区网络（CloudCampus）解决方案，应用前沿的有线和无线技术，加持大数据、AI和云技术，以业务为中心，构建万物互联、业务无忧和可平滑演进的园区网络，使能行业数字化转型。

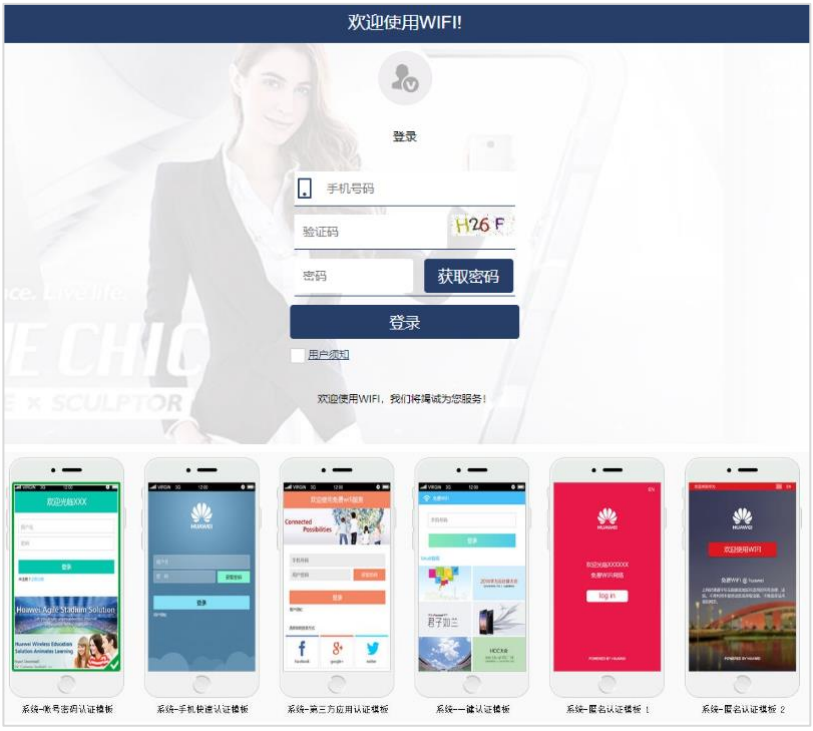
本实验将指导读者开发第三方认证应用，调用华为 iMaster NCE-Campus 的第三方认证 API，实现网页重定向、访客信息读取、访客上\下线管理等。本课程您会学习：

- 对接 iMaster NCE-Campus 的第三方认证接口。
- 华为授权 API 方式用户上下线流程。

3.1.1 第三方认证介绍

第三方认证主要应用于商业 Wi-Fi 场景。用户在商场酒店、机场地铁、企业来访等场景通过 Wi-Fi 访问互联网时，网络提供方需要对接入网络的访客进行用户认证，同时提供宣传、推荐及营销等功能。访客通过认证后才被允许接入 Wi-Fi 使用网络。

本实验中开发者为接入访客提供认证 Portal 页面，并调用华为 iMaster NCE-Campus 第三方认证 API，从而实现认证、计费、用户分析、市场营销等服务。



3.2 实验介绍

3.2.1 组网说明

本实验组网包含四个对象，终端、CloudAP、iMaster NCE 和第三方认证应用。其中终端、CloudAP、iMaster NCE 可由开发者社区提供沙箱环境，读者需编写位置应用代码。第三方认证应用代码推荐使用华为云 CloudIDE 运行。

3.2.2 实验步骤

本实验步骤如下：

1. 环境准备：准备 IDE 和实验沙箱环境。
2. 第三方认证应用开发：使用 Flask 框架编写认证应用代码。
3. 结果验证：配置 SSID 和用户接入，验证结果。

3.3 环境准备

3.3.1 IDE 准备

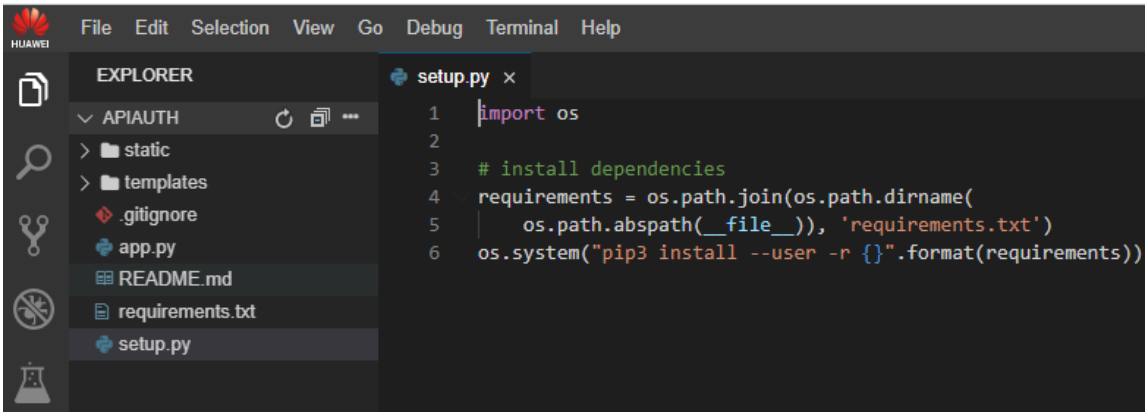
本章节完整代码已上传到华为云 DevCloud。读者可以使用华为 CloudIDE 在线编程或将代码下载至本地编程：https://devcloud.cn-north-4.huaweicloud.com/codehub/project/68494a8ad06b4eea9a1c3f18be115161/codehub/578588/home。

1.访问 CodeHub 控制台，新建项目和云上代码仓库。

更多操作请参考 CodeHub 指南，https://support.huaweicloud.com/codehub/index.html。

2.将完整代码 Fork 到仓库中，点击 CloudIDE 即可打开代码。

3.CloudIDE 界面显示如下：



3.3.2 沙箱预约

开发者社区提供华为数通解决的沙箱体验，本实验申请智简园区网络沙箱。

沙箱提供两种体验方式，快速体验和 Demo 调测：

- 快速体验使用公网接入实验室，适用于 CloudIDE 开发代码；
- Demo 调试使用 VPN 接入实验室，适用于本地 IDE 开发代码。

快速体验下选择“控制器、设备及终端”和时长，点击申请，即可获得环境登录信息。

“租户账号”为用户 NCE 页面登录账户。

“北向账号”为位置定位应用与 NCE 交互的账号。组网说明中 1-3 步骤。

“北向 URL”为 iMaster NCE 登录地址，点击“前往体验”可跳转。

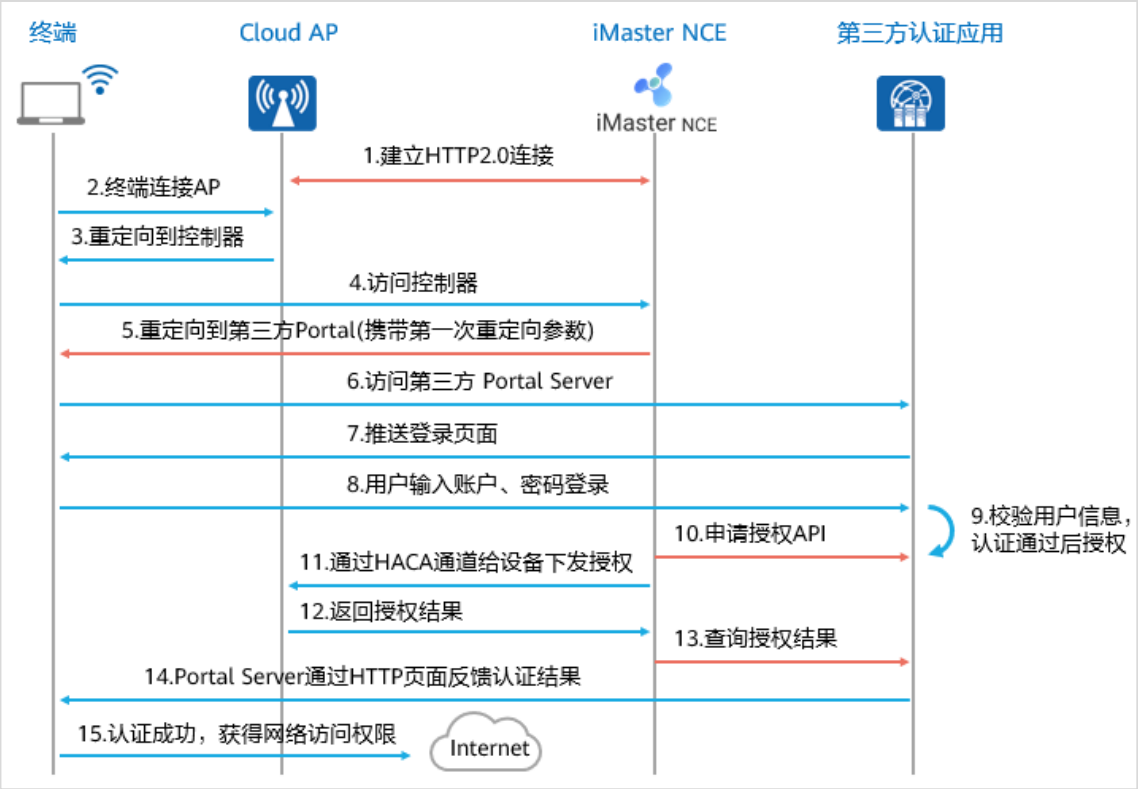
“密码”为本次申请的沙箱环境的登录密码。

至此，环境准备完成。

3.4 第三方认证应用开发

本章节您将尝试编写一个认证应用 Demo，与 iMaster NCE-Campus 对接实现 Wi-Fi 用户 Portal 认证。

本实验各组件具体交互流程如下：



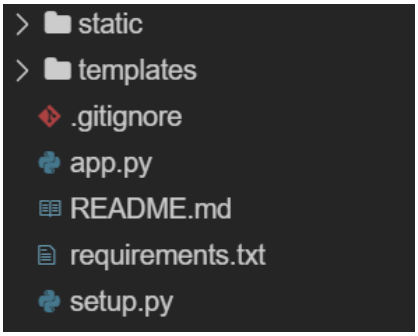
认证服务器需要实现三项功能：

- Portal 服务器(推送认证页面，接收用户登录信息，返回登录结果)；
- 解析、存储和校验用户信息；
- 调用 iMaster NCE API 给予终端权限。

3.4.1 代码目录结构

本应用 Demo 基于 Python Flask 框架编写。

代码结构如下：



- static 目录下存放图片和 CSS 资源。
- templates 目录存放 HTML 页面。
- app.py 为本应用的主要逻辑。
- setup.py 作为为安装应用的依赖包。
- requirements.txt 存放本次实验需要的依赖包路径。

3.4.2 安装 iMaster NCE-Campus SDK

代码仓库中 requirements.txt 已写入依赖包，完整信息如下：

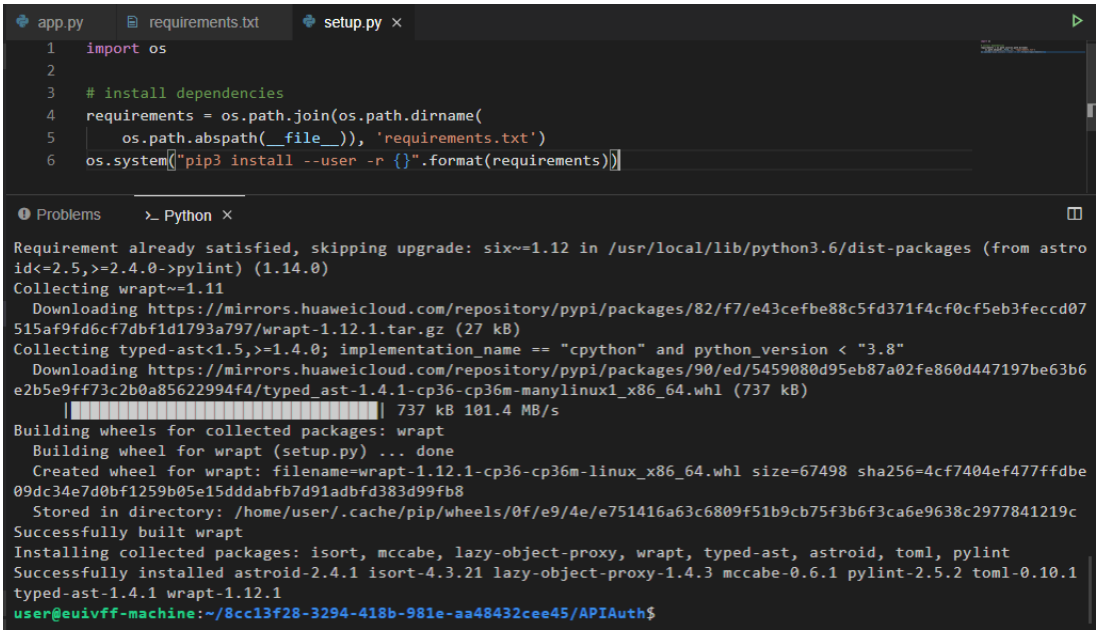
```
certifi >= 14.05.14
six >= 1.10
python_dateutil >= 2.5.3
setuptools >= 21.0.0
urllib3 >= 1.15.1
flask >= 1.1.1
requests >=2.22.0

-e git+https://codehub.devcloud.cn-north-
4.huaweicloud.com/campusoooo1/CloudCampusPythonSDK.git@master#egg=cloudcampus
```

更多智简园区资源信息请参考 https://devzone.huawei.com/apistudio/sample/campus/apiSdk.html。

setup.py 包含安装代码。本地获取 SDK 请参考“智简园区网络”>“资源下载”。

```
import os
requirements = os.path.join(os.path.dirname(
      os.path.abspath(__file__)), 'requirements.txt')
os.system("pip3 install --user -r {}".format(requirements))
```

运行 setup.py 代码安装 SDK（CloudIDE 在 setup.py 中右键单击“Run Python File in Terminal”或点击运行即可）。



安装成功。

3.4.3 初始化 API 客户端

使用 ApiClient 类来初始化您的 SDK。您需要配置北向 API 使用的端口号（18002），iMaster NCE-Campus 控制器的 IP 以及北向帐号和密码。

```
if __name__ == "__main__":

    # init api
    tenantName = ' demo15@north.com'
    tenantPwd = Sky7xe1Mx@''
    host = '139.9.213.72'
    port = '18002'
    config = Configuration(host, port, tenantName, tenantPwd)
    api_client = ApiClient(config)
```

请注意此处信息需要和沙箱环境保持一致。

3.4.4 解析用户信息

解析用户信息前需要首先配置 Flask 应用，然后用户请求将被重定向到此应用，最后解析用户终端信息。

1.配置和启动 Flask 应用（app.py）

```
app = Flask(__name__)

if __name__ == "__main__":

        # Flask APP  启动
    app.run(
        host="0.0.0.0",
```

```
        port=8899
    )
```

2.解析用户请求 request 中的信息（app.py）

```
@app.route('/', methods=['GET', 'POST'])   #用户的登录主页面和 HTTP 方法
def index():

    # 从参数中解析用于调用 API 的参数
    uaddress = request.args.get('uaddress')
    umac = request.args.get('umac')
    ssid = request.args.get('ssid')
    apmac = request.args.get('apmac')
    node_ip = request.args.get('nodeIp')
```

参数从上至下分别为认证终端 IP、认证终端 MAC 地址、认证终端接入的 SSID、认证终端接入的 AP 的 MAC 地址和 iMaster NCE 的 IP 地址。

更多参数详细信息请参考开发指南。

3.4.5 用户登录

1.在用户登录的主页面判断 HTTP 的方法。

如果是 GET 方法，则返回登录页面；如果是 POST 方法，进入授权流程。

```
if request.method == 'GET':
        return render_template('login.html')     #如果 HTTP 方法为 GET，返回 login.html 页面
    else:
        username = request.form.get('username')
        # print(username)
        password = request.form.get('password')
        # print(password)
        #本实验设置的用户账户密码
        if username == '123' and password == '123':
        # 授权用户...
```

- 如果用户在主界面 HTTP 操作为 GET，则返回 template 目录下的 login.html 页面。
- 如果 HTTP 操作为 POST，且用户输入的账号、密码均为 123，将执行授权用户操作。

本次实验使用的默认账号密码为“123/123”，开发者可自行修改或对接数据库。

2. 调用授权终端用户 API，并将用户信息发往 success 页面，用于在用户下线调用（app.py）。

```
    # 获取授权 API
    client_user_manager_api = ClientUserManagerApi(api_client)
    # 授权终端用户
    try:
        # 新建用户信息
        user_dto = UserAuthorizationInputDto(device_mac=apmac, terminal_ip_v4=uaddress,
terminal_mac=umac, ssid=ssid, node_ip=node_ip, user_name=username)
```

```
    except ValueError as e:
        # 捕获异常
        print('EXCEPT :    ' + str(e))
        return render_template('login.html', err_msg = 'request parameter')
    # 调用授权 API 并打印结果
    auth_output = client_user_manager_api.user_authorization(user_dto)
    print(auth_output)
```

新建用户信息 user_dto。参考 API 手册可知 HTTP 请求 Body 携带 UserAuthorizationInputDto 的详细参数。具体方法可以参考其中的“代码示例”Python。

3.调用查询授权结果 API，确认授权成功，并重定向至 success 页面（app.py）。

```
    time.sleep(3)   #等待三秒查询授权结果
    query_auth_output = client_user_manager_api.get_authorizationresult(auth_output.psessionid,node_ip)
    print(query_auth_output)

    # 查询失败 返回 error
    if query_auth_output.errmsg != 'true':
        return render_template('login.html', err_msg = 'query auth')

    # 查询成功 重定向到 success 页面
    return redirect(url_for('success', uaddress = uaddress, umac=umac, ssid=ssid, apmac=apmac, nodeIp=node_ip,
psessionid=auth_output.psessionid, username=username))
```

调用查询 API，具体方法参考 API 手册。

如果查询成功，用户页面将重定向到/success 的 URL。

4.配置 success 页面。

用户请求携带参数，访问 success 页面。

```
@app.route('/success', methods=['GET', 'POST'])
def success():
    if request.method == 'GET':
        data = request.url.lstrip(request.base_url)
        print(data)
        # 参数保留 下线时再次使用
        return render_template('success.html', querystring = '/logout' + data)
```

返回给用户 template 目录下 success.html。

3.4.6 用户下线

用户主动下线将调用下线 API。

1.解析 Cookie 中存储的用户数据（app.py）。

```
@app.route('/logout', methods=['GET', 'POST'])
def logout():
    # 解析请求参数
    uaddress = request.args.get('uaddress')
    umac = request.args.get('umac')
```

```
    apmac = request.args.get('apmac')
    node_ip = request.args.get('nodeIp')
    psessionid = request.args.get('psessionid')
    username = request.args.get('username')
    ssid = request.args.get('ssid')
```

2. 调用用户下线 API，并重定向至登录界面（app.py）。

```
    # 获取下线 API
    client_user_manager_api = ClientUserManagerApi(api_client)

    # 强制用户下线
    third_user_info = ThirdUserInfoData(device_mac=apmac, terminal_ip_v4=uaddress, terminal_mac=umac,
node_ip=node_ip, psessionid=psessionid, user_name=username)
    print(third_user_info)
    model = client_user_manager_api.cut_user(CutUserInputDto([third_user_info]))

    print(model)
  # 携带参数 重定向到登录页面
    return redirect(url_for('index', uaddress = uaddress, umac=umac,ssid=ssid,apmac=apmac,nodeIp=node_ip))
```

ClientUserManager 提供强制用户下线北向接口。其 Python 方法为 cut_user，具体示例请参考 API 手册。

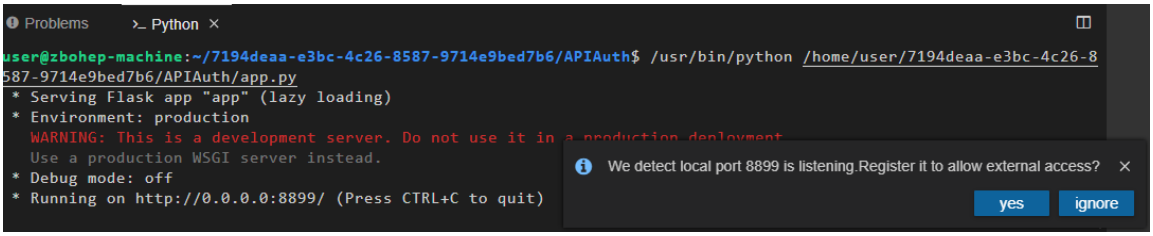
3.5 结果验证

本实验结果验证首先启动第三方认证服务，然后在 iMaster NCE-Campus 配置 Portal 认证对接，最后用户终端无线接入验证。

3.5.1 启动认证应用

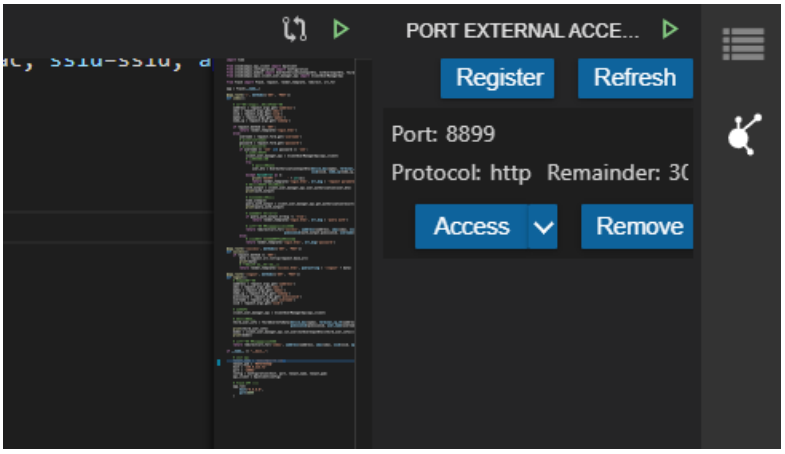
1.启动服务器。

CloudIDE 中运行 app.py 文件。

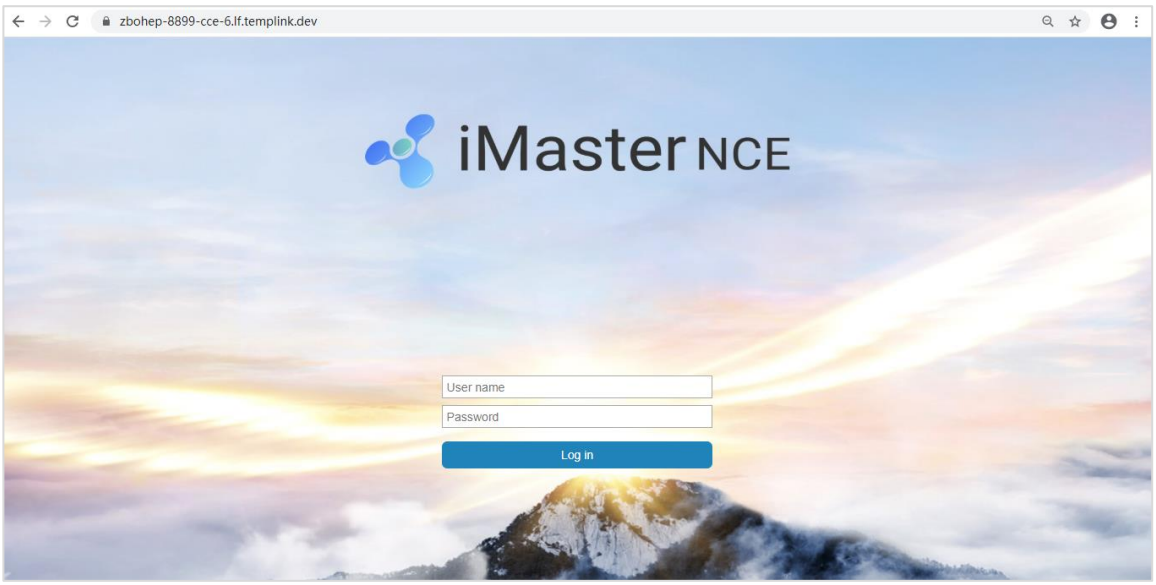


2.将程序映射至公网(本地 IDE 调试不需要)。

点击“yes”后在窗口右上角出现 PORT EXTERNAL ACCESS。单击“Access”后进入 Portal 的登录界面。注册端口默认为 8899。



3.查看 Portal 登录页面。



页面 URL 本例为 https://zbohep-8899-cce-6.lf.templink.dev/。

3.5.2 配置 Portal 认证对接

在 iMaster NCE-Campus 上配置 Portal 认证对接。

3.5.2.1 配置 SSID

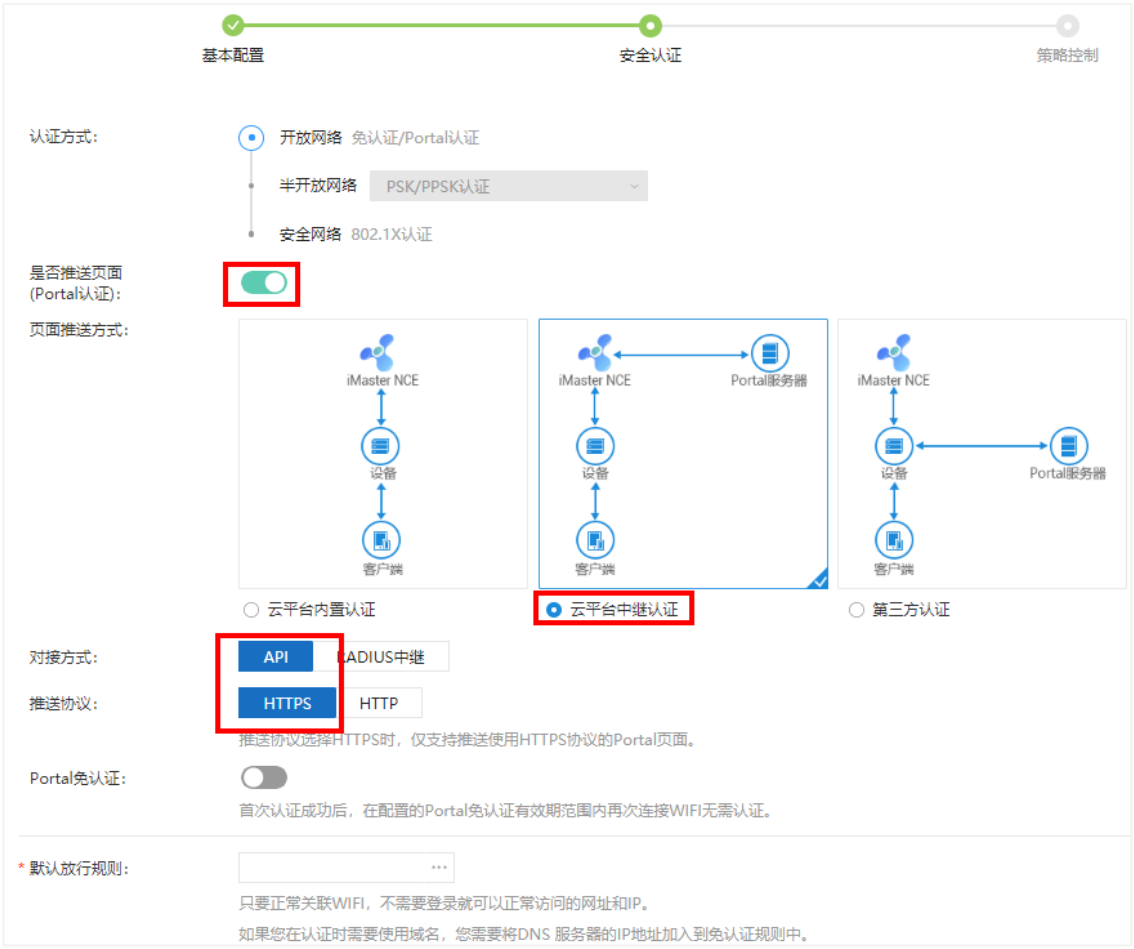
1.创建 SSID。

根据沙箱信息登录后，在主菜单选择“配置>物理网络>站点配置>AP”。

2.选择"创建"，输入 SSID 名称，选择网络连接方式为“NAT”，单击“下一步”。

3.配置终端用户使用 SSID 接入网络时的认证方式。

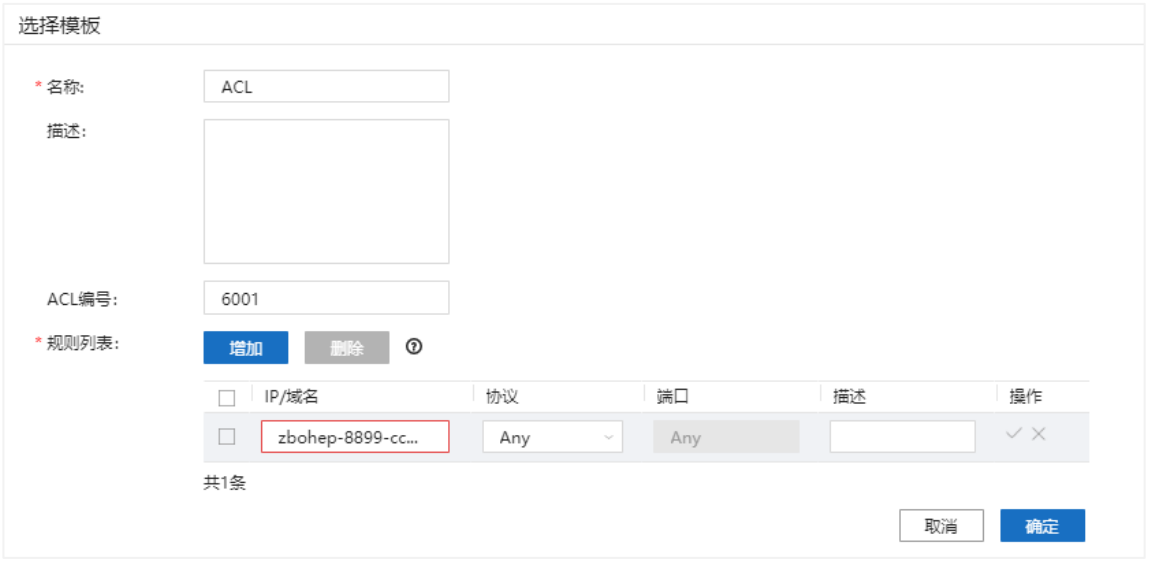
设置“认证方式”为“开放网络”，“是否推送页面（Portal 认证）”为“ON”，“页面推送方式”为“云平台中继认证”，“对接方式”为“API”，推送协议为“HTTPS”。



4.配置默认放行规则。

单击“默认放行规则”右侧的“选择模板”，在弹出的“选择模板”对话框中单击“创建”，设置 ACL 模板规则。

复制“启动认证应用”中的域名，将域名加入规则列表。注意将协议部分(http/https)删除，只保留域名部分。



勾选逃生策略后，单击“下一步”，随后单击“确定”，完成 SSID 配置。

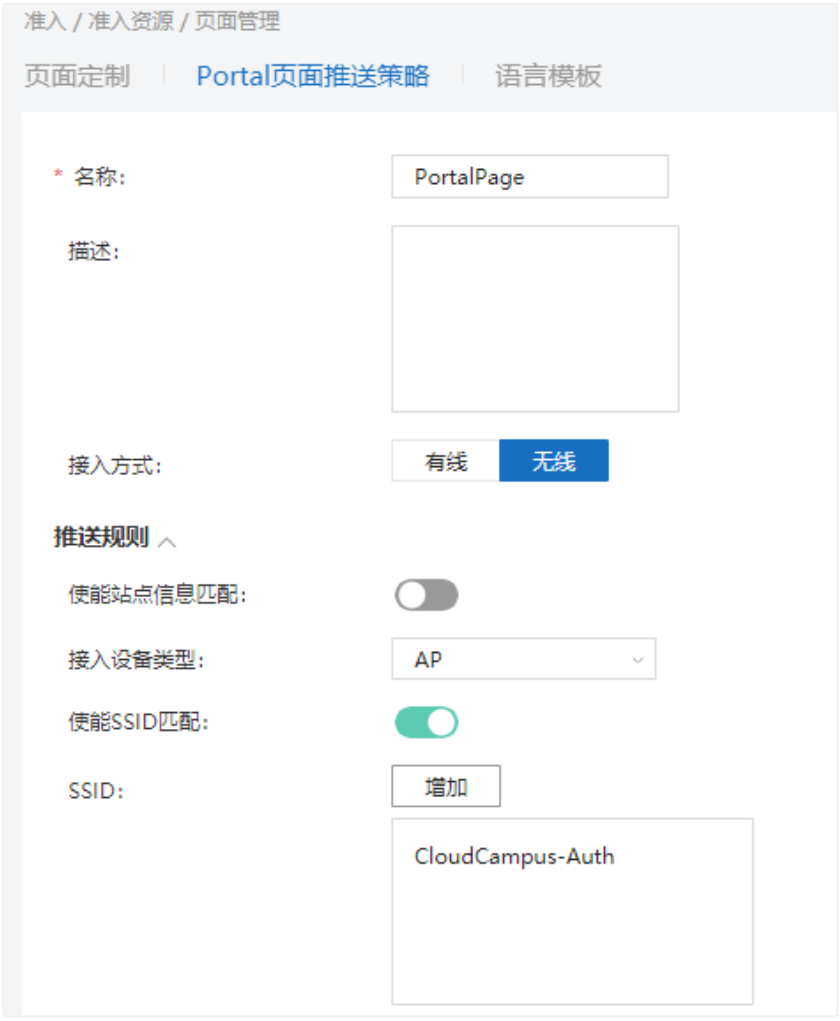
3.5.2.2 配置 Portal 页面推送策略

云平台中继 API 认证模式下，用户需配置 Portal 推送策略。当终端关联 WI-FI 后，根据 Portal 推送策略给终端用户推送指定的 Portal 页面。

1. 设置 Portal 页面推送名称和关联的 SSID。

在主菜单中选择“准入 > 准入资源 > Portal 页面推送策略”。

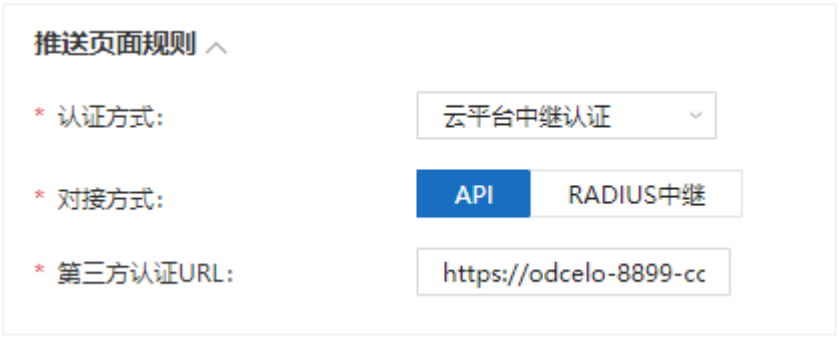
单击“创建”，选择“设备类型”为 AP，配置 Portal 页面推送策略。



选择策略关联的 SSID。

2.配置对接方式和认证服务器。

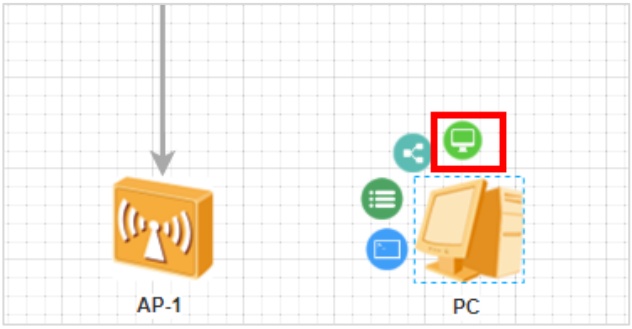
选择“云平台中继认证”，然后在第三方认证 URL 复制 CloudIDE 启动服务后生成的 URL。



最后点击应用。至此对接配置完成。

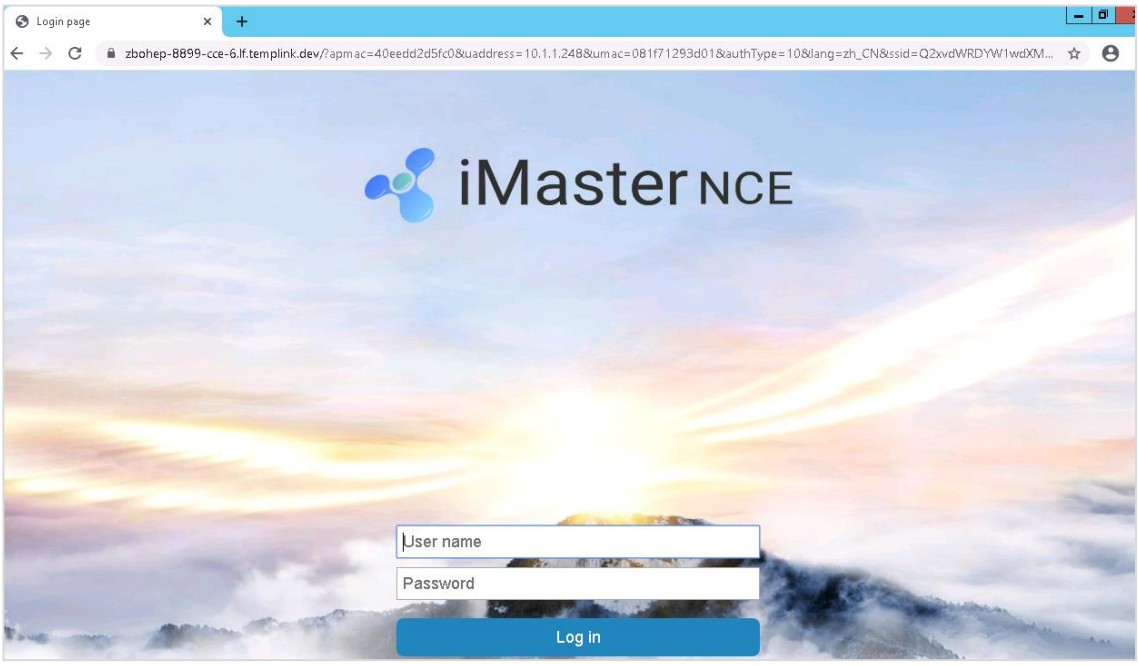
3.5.3 用户终端接入验证

在实验室预约界面，单击“进入拓扑”，选择 PC 上的远程按钮，进入远程实验室 PC。

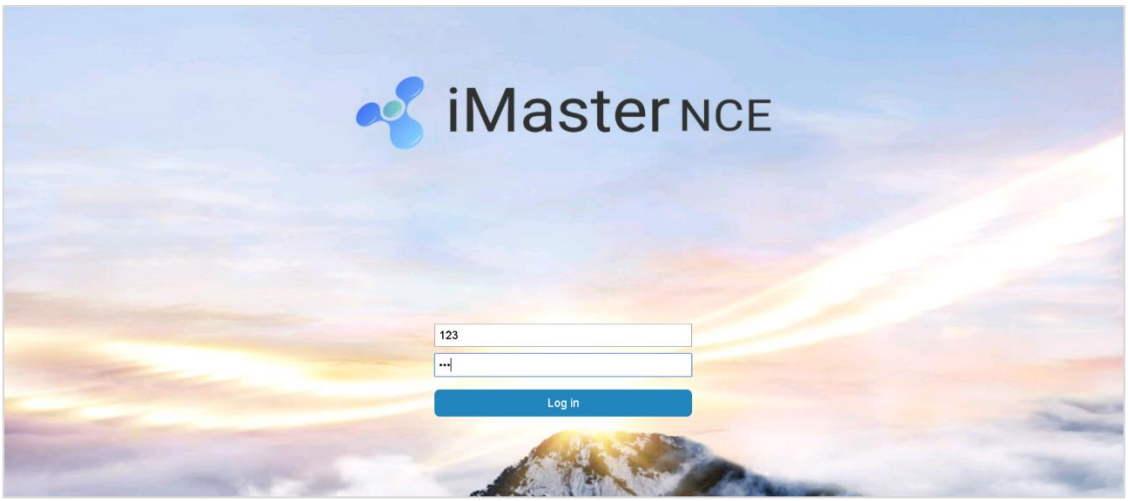


进入远程实验室 PC 后，选择 Wi-Fi，连接上一步配置完成的 SSID。

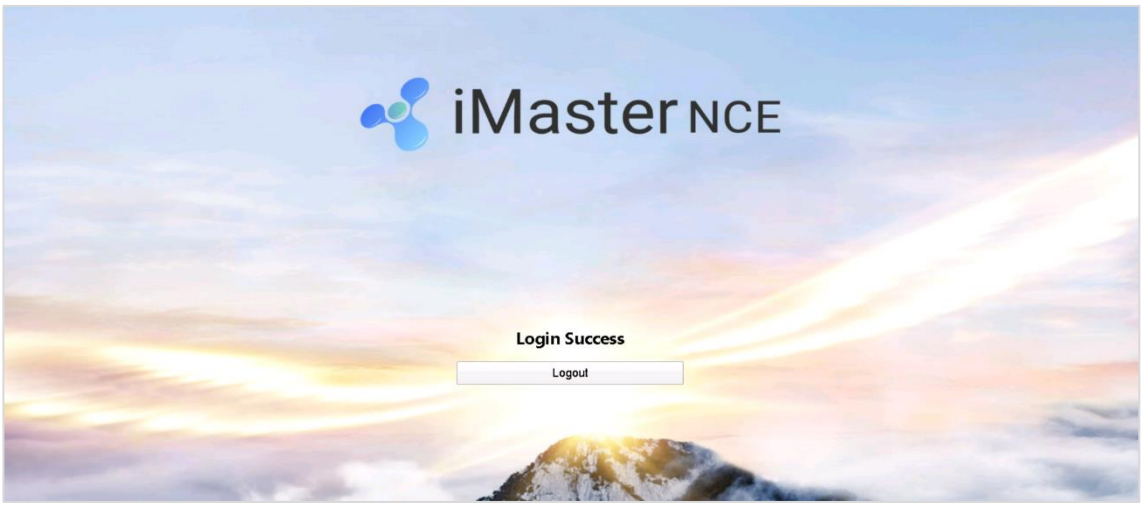
PC 接入 Wi-Fi 信号，打开浏览器，访问任意地址，会自动跳转到本次教程的服务器地址。



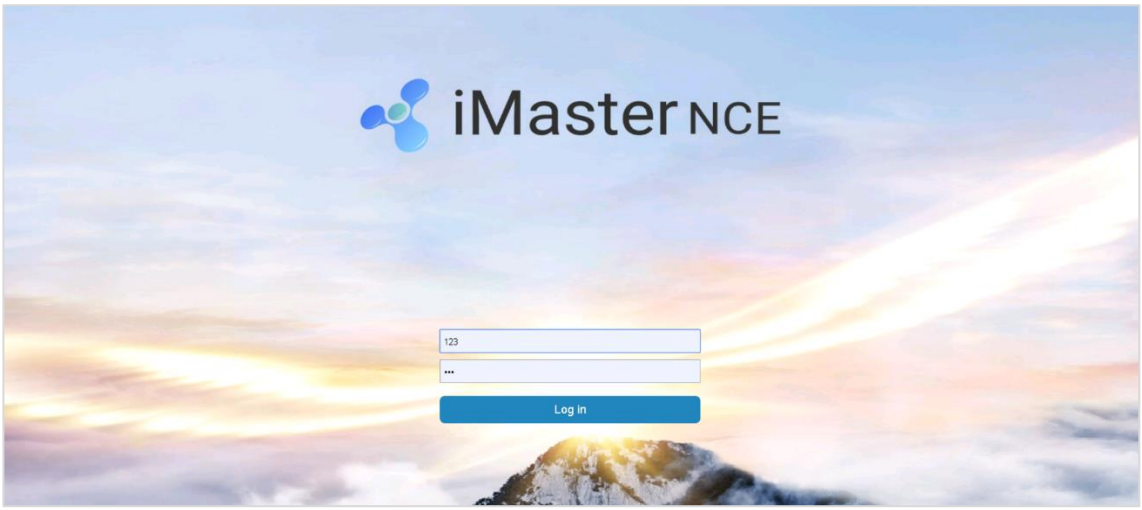
输入用户名、密码，单击“Log in”进行登录（代码中默认用户名和密码均为“123”）。



登录成功,终端能够成功访问互联网。



单击“Logout”，注销后返回登录页面。



至此，本章节第三方认证实践结束。

4 智简数据中心业务发放实践

4.1 实验背景

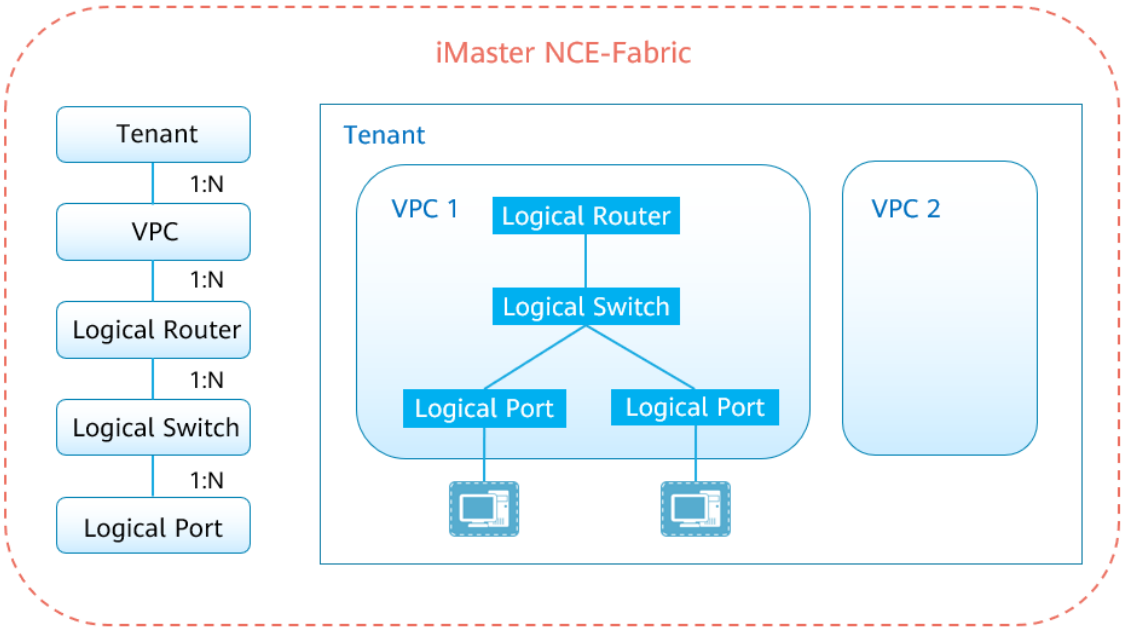
华为智简数据中心网络（CloudFabric）解决方案，为客户提供智能超宽、智能联接和智能运维的下一代数据中心网络，帮助企业从数据中挖掘智慧，加速数字化转型和推动数字经济发展。

本实验将指导读者开发使用 Python 调用华为 iMaster NCE-Fabric 的北向 API，实现 VPC 网络业务发放。本课程您会学习：

- iMaster NCE-Fabric 逻辑网络模型。
- iMaster NCE-Fabric Web UI 发放业务。
- 使用 Python 调用相关 API 发放业务网络。

4.1.1 业务发放介绍

业务发放是指基于 iMaster NCE-Fabric 强大的北向开放 API 能力，实现网络快速部署。

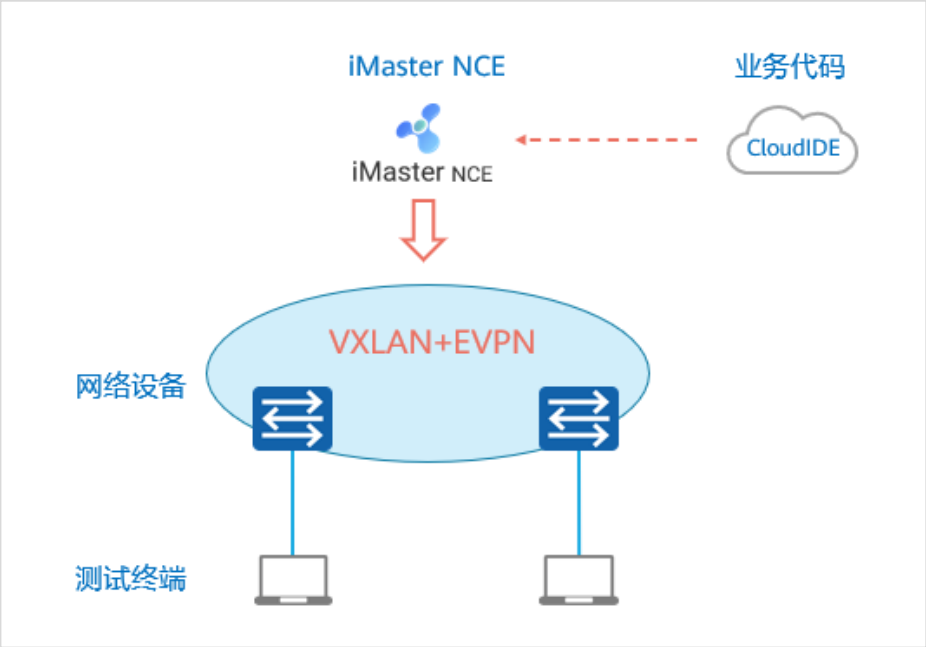


iMaster NCE-Fabric 支持多租户（Tenant），租户之间彼此隔离。一个租户内有多种逻辑网络对象由租户管理员自行编排：

- VPC（Virtual Private Cloud）：VPC 是一个隔离的安全域，一般用于满足不同业务的网络隔离需求。
- 逻辑路由器（Logical Router）：逻辑路由器顾名思义是逻辑的网关，是用于连接内部不同子网和内部与外部网络的逻辑网络对象。
- 逻辑交换机（Logical Switch）：逻辑交换机顾名思义是逻辑的二层交换设备，用于实现同一个网段内业务的二层通信。
- 逻辑端口（Logical Port）：逻辑端口指业务的接入端口。这个端口可以位于不同网络设备上，接入逻辑交换机二层互通。底层原理由 VXLAN 和 EVPN 实现。

4.2 实验介绍

4.2.1 组网说明



本实验组网包含四个对象，测试终端、网络设备、iMaster NCE 和业务代码。其中终端、网络设备、iMaster NCE 可由开发者社区提供沙箱环境。读者需编写业务代码，推荐使用华为云 CloudIDE 运行。

4.2.2 实验步骤

为了帮助读者更好掌握 iMaster NCE-Fabric 原理，本实验分为两个部分：基于 Web UI 发放网络业务和使用 Python 代码发放网络业务。

1.基于 Web UI 发放业务

掌握在 iMaster NCE-Fabric 页面操作，实现业务下发包括：

- 环境准备
- 创建租户
- 创建 VPC
- 创建逻辑路由器
- 创建逻辑交换机
- 创建逻辑端口

2.使用 Python 代码发放业务

- 编写业务发放代码

4.3 环境准备

4.3.1 IDE 准备

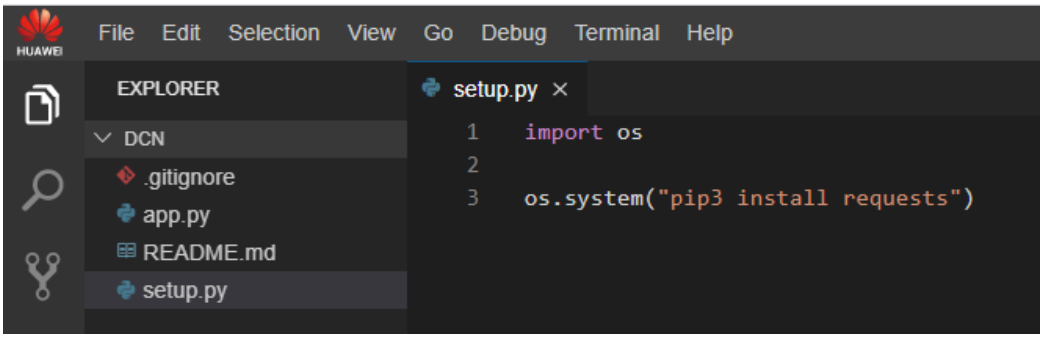
本章节完整代码已上传到华为云 DevCloud。读者可以使用华为 CloudIDE 在线编程或将代码下载至本地编程：https://devcloud.cn-north-4.huaweicloud.com/codehub/project/fa019fde75e740d58870b40f551bf749/codehub/625828/home。

1.访问 CodeHub 控制台，新建项目和云上代码仓库。

更多操作请参考 CodeHub 指南，https://support.huaweicloud.com/codehub/index.html。

2.将完整代码 Fork 到仓库中，点击 CloudIDE 即可打开代码。

3.CloudIDE 界面显示如下：



4.3.2 沙箱预约

开发者社区提供华为数通解决的沙箱体验，本实验申请智简数据中心网络沙箱。

沙箱提供两种体验方式，快速体验和 Demo 调测：

- 快速体验使用公网接入实验室，适用于 CLoudIDE 开发代码；
- Demo 调试使用 VPN 接入实验室，适用于本地 IDE 开发代码。

快速体验下选择时长，点击申请，即可获得环境登录信息。

“租户账号”为用户 NCE 页面登录账户。

“密码”为本次申请的沙箱环境的登录密码。

至此，环境准备完成。

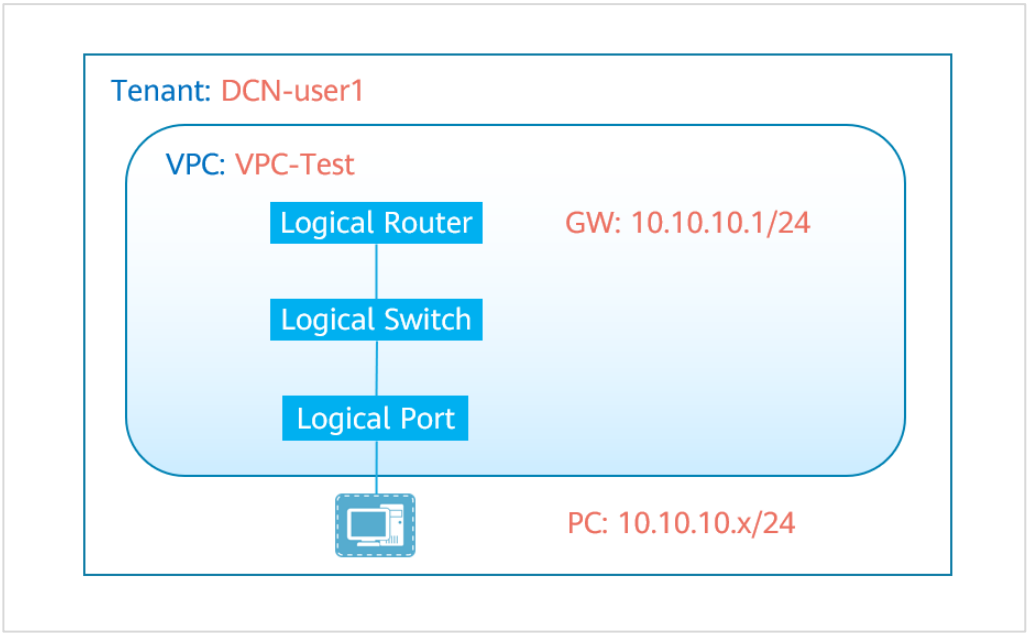
4.4 基于 Web UI 的业务发放

本小结您将会使用 iMaster NCE-Fabric Web 页面进行业务下发，具体步骤为：

1.创建租户。

2.部署租户网络业务。

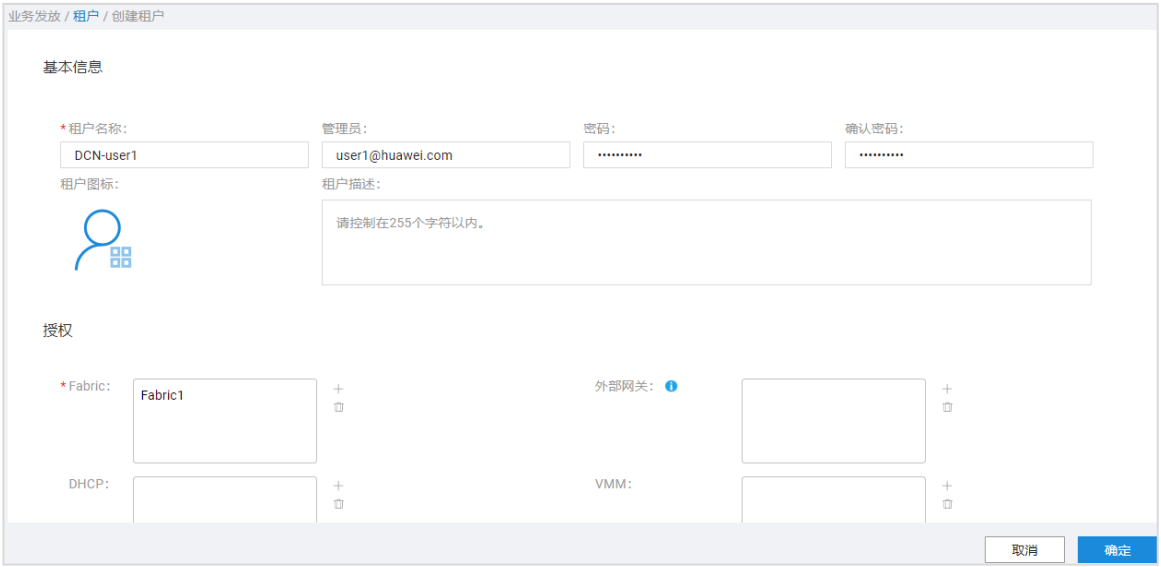
最终完成创建租户“DCN-user1”，创建 VPC“VPC-Test”，并创建网段 10.10.10.0/24，网关为 10.10.10.1/24。实现测试终端到联通网关。



4.4.1 创建租户

“沙箱预约”步骤中申请的账户“admin”为系统管理员账户。使用此账户登录 iMaster NCE-Fabric。

选择“业务发放”>“租户”，点击“创建租户”。



填写租户信息：

- 租户名称
- 管理员：租户管理员账户

- 密码：租户管理员密码
- Fabric：Fabric 是 iMaster NCE-Fabric 上创建的对象。将一组 Spine 和 Leaf 设备添加到 Fabric 后，Fabric 代表的物理网络可提供给多个租户同时使用。本例中已预先创建好“Fabric1”，并已添加设备。

点击“确认”后，租户创建成功。

注销“admin”账户后，使用租户管理员账户登录验证。首次登录需修改登录密码。

登录成功。

4.4.2 部署租户网络业务

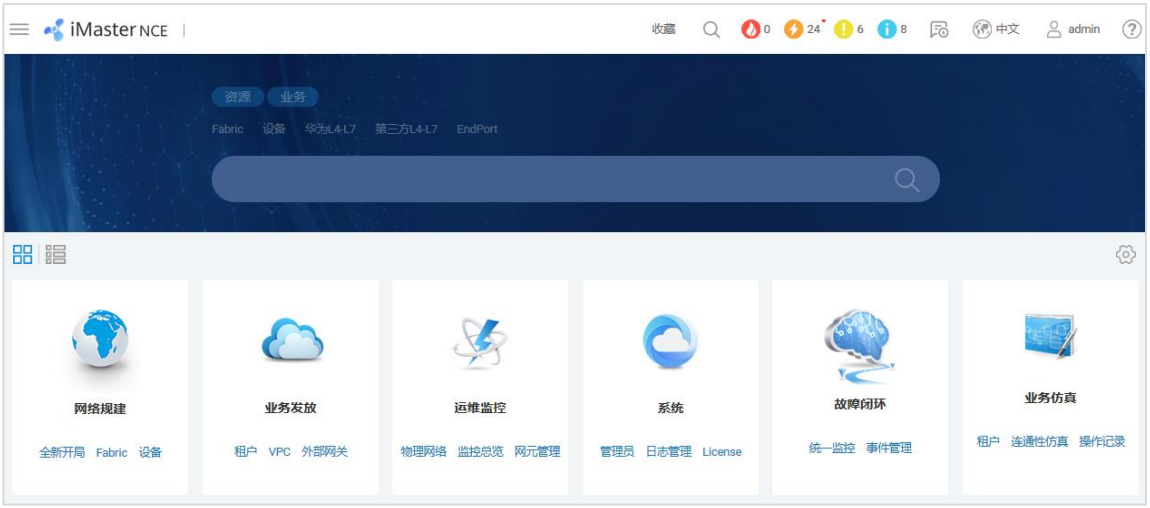
使用租户管理员账户部署租户业务，本实现部署单 VPC 网络，实现步骤为：

- 创建 VPC
- 创建逻辑路由器
- 创建逻辑交换机
- 创建逻辑端口

本案例物理交换机的 10GE1/0/15 连接测试终端。

4.4.2.1 创建 VPC

重新使用“admin”系统管理员登录 iMaster NCE-Fabric。

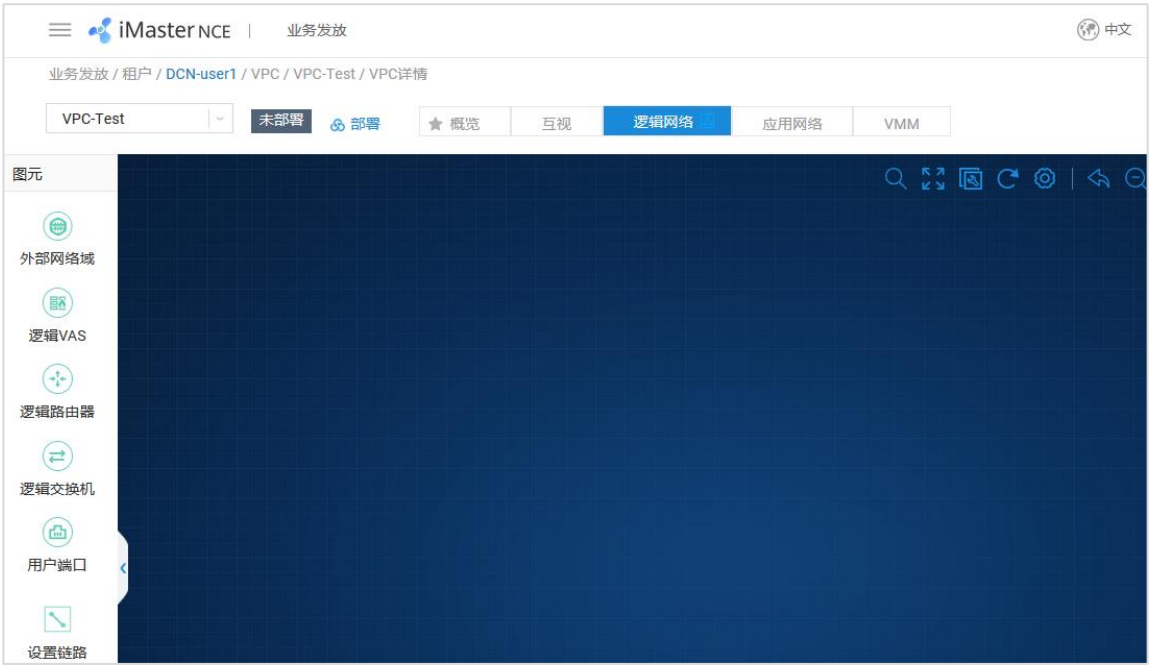


点击“业务发放”>“租户”。点击租户头像进入租户视图。

选择“VPC”，点击“创建”。

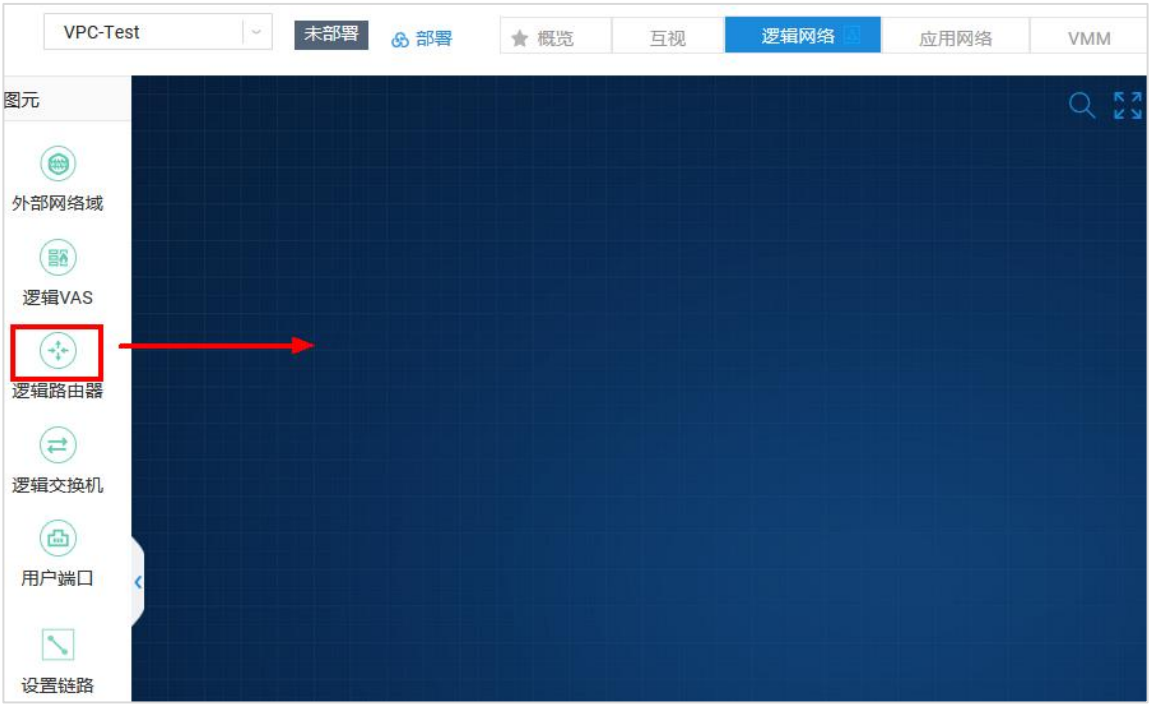
输入 VPC 参数，点击确认。

自动跳转到 VPC 编排页面。



4.4.2.2 创建逻辑路由器

从左侧菜单栏拖动“逻辑路由器”到右侧窗口。



右侧弹出配置窗口。填写逻辑路由器名称和选择部署的 Fabric。本例为“Router1”和“Fabric1”。

为逻辑路由器添加子网信息 10.10.10.0/24，网关为 10.10.10.1。点击“增加”，填写信息。

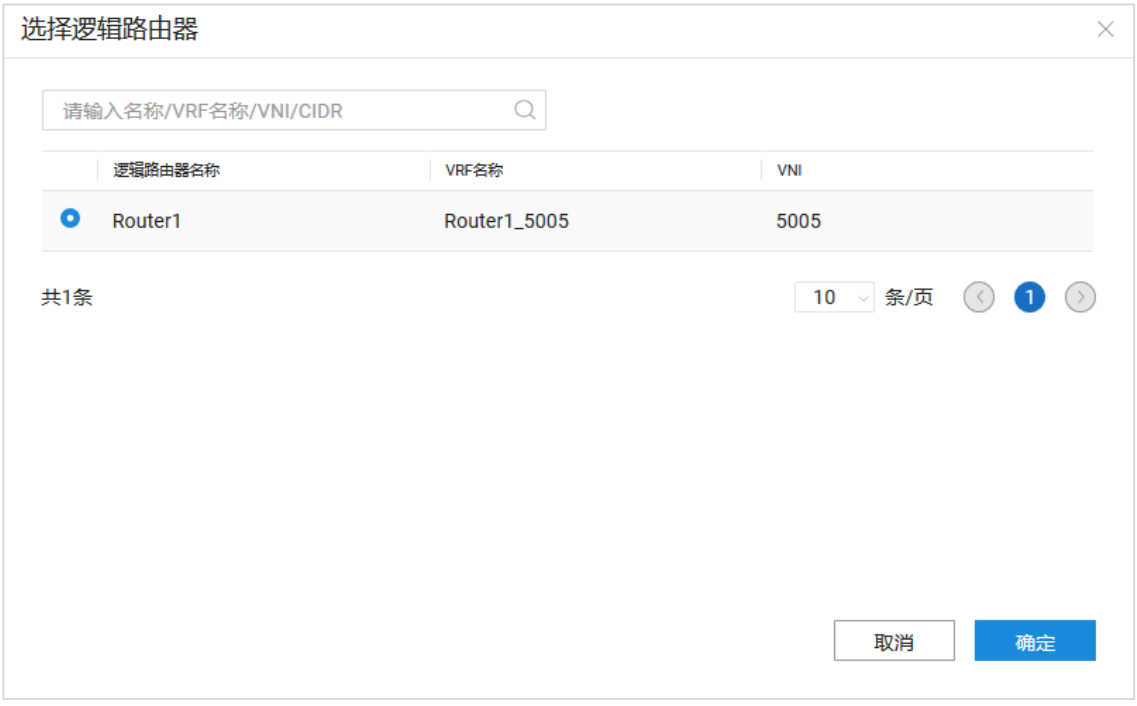
点击“确定”，完成子网创建。

点击“确定”，完成逻辑路由器创建。其名称变为“Router1”。

4.4.2.3 创建逻辑交换机

同样拖动“逻辑交换机”到右侧区域。填写逻辑交换机名称，本例为“Switch1”。

为逻辑交换机选择关联的逻辑路由器。点击“选择”，填出窗口。



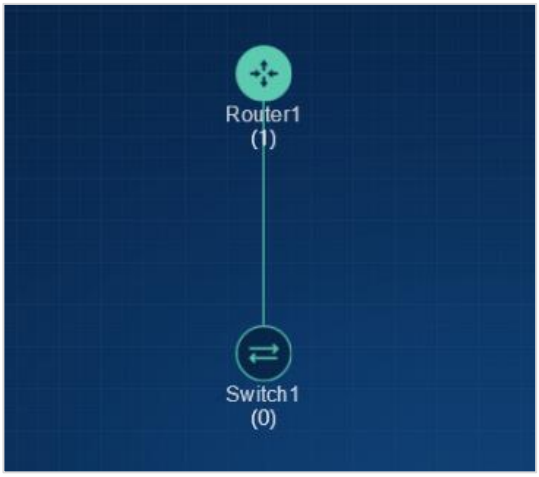
本例选定逻辑路由器“Router1”。点击“确定”，确认关联。

选择“编辑子网”，将逻辑交换机关联子网。

在弹出的窗口中选择子网“10.10.10.0/24”，并点击“确定”。

点击“确定”创建逻辑交换机。

至此，完成逻辑交换机的创建与配置。

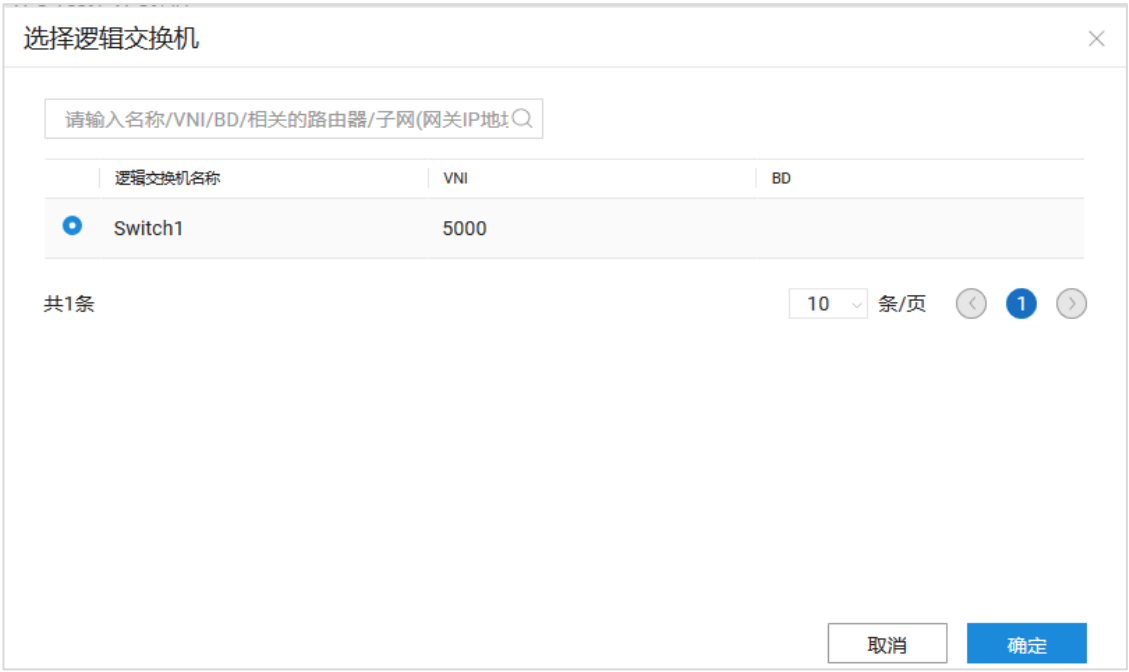


4.4.2.4 创建逻辑端口

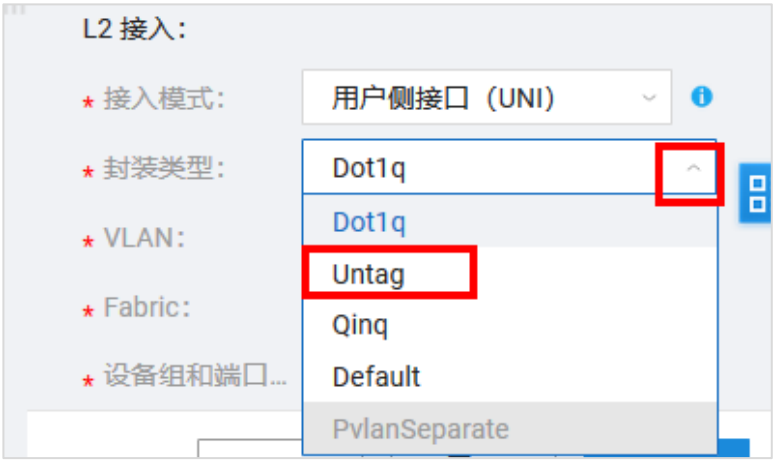
拖动“逻辑端口”到右侧，弹出配置窗口。

填写逻辑端口名称，本例为“Port1”。

点击“选择”关联逻辑交换机。在弹出窗口中选择“Switch1”，并点击确定。



配置“L2 接入”。选择用户侧接口，在封装类型的下拉菜单中选择“Untag”。因为本例中测试终端为 PC，发出的流量不携带 VLAN Tag。

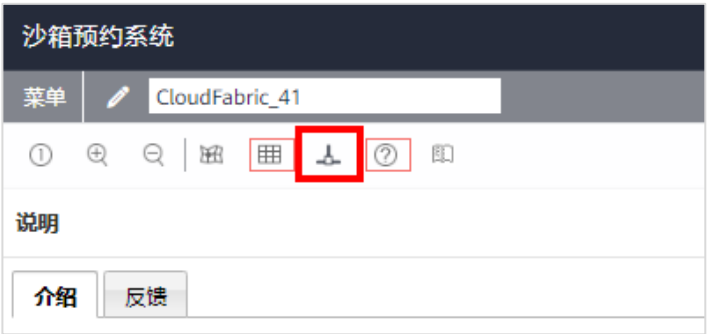


选择“Fabric1”后，点击“选择”以选定测试终端连接的物理接口。

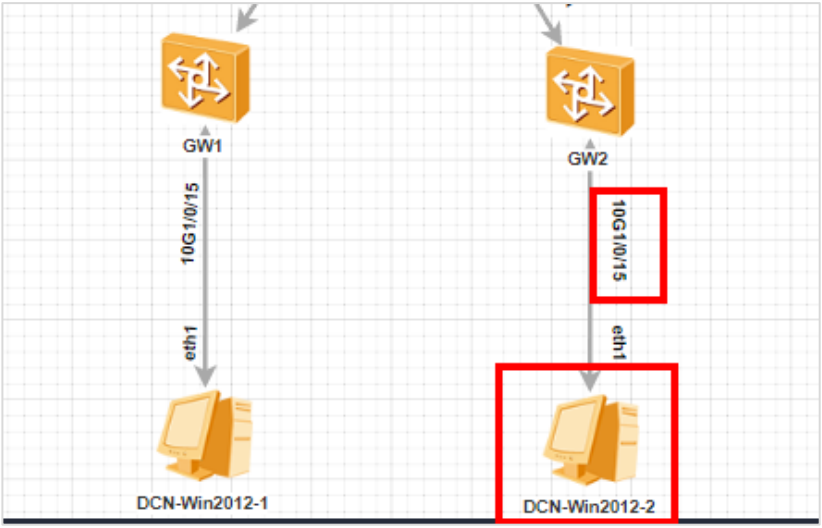
在弹出窗口中选择测试终端真实接入的端口。

在“沙箱预约”中页面点击“进入拓扑”可查看物理接口。

点击如下图标，拓扑中将显示接口。



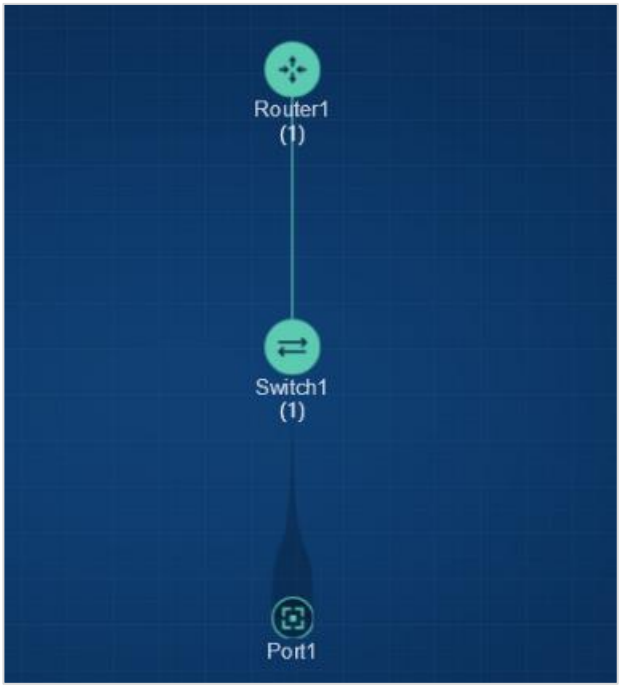
“DCN-Win2012-2”作为测试终端，接入 GW2 的 10GE1/0/15 接口。



在创建逻辑端口窗口中选择此接口。

点击“确认”完成创建。

现在逻辑网络拓扑显示如下：

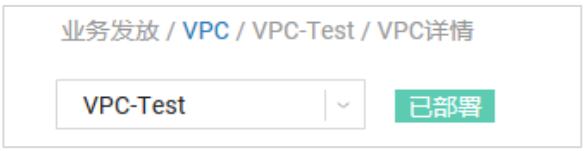


4.4.2.5 部署 VPC 网络

VPC 需要被部署，才能应用在物理网络上。

在 VPC 编排界面的右上角。

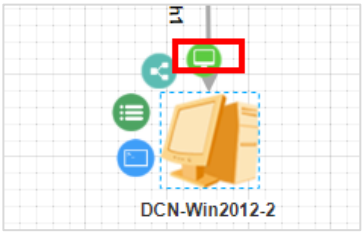
部署成功。



4.4.3 结果验证

1.登录测试终端。

在沙箱拓扑中，点击登录测试终端。



Ping 测试网关连通性。

```
C:\Users\Administrator>ping 10.10.10.1

正在 Ping 10.10.10.1 具有 32 字节的数据:
来自 10.10.10.1 的回复：字节=32 时间=1ms TTL=254
来自 10.10.10.1 的回复：字节=32 时间=1ms TTL=254
来自 10.10.10.1 的回复：字节=32 时间=1ms TTL=254
来自 10.10.10.1 的回复：字节=32 时间=1ms TTL=254

10.10.10.1 的 Ping 统计信息:
    数据包: 已发送 = 4，已接收 = 4，丢失 = 0 (0% 丢失)，
```

2.删除 VPC 和租户。

在“业务发放”>“VPC”页面删除 VPC-Test。

选择“业务发放”>“租户”，删除租户 DCN-user1。

此时再在终端上测试连通性。

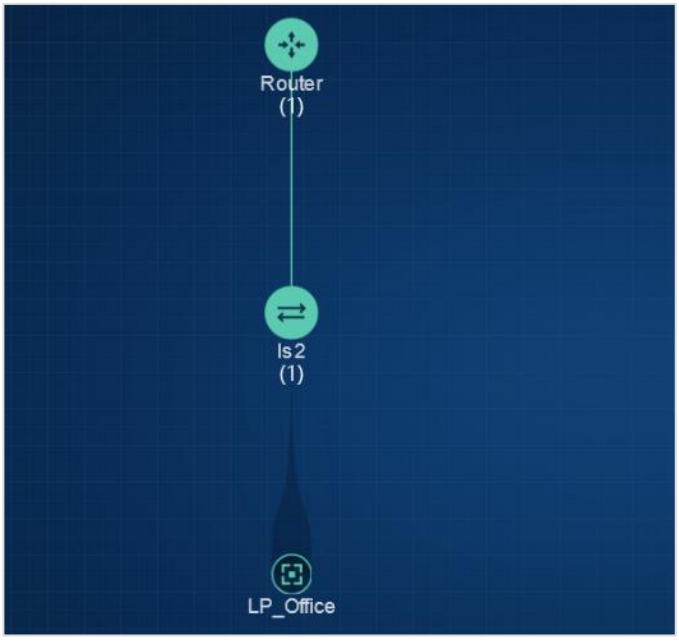
```
C:\Users\Administrator>ping 10.10.10.1

正在 Ping 10.10.10.1 具有 32 字节的数据:
来自 10.10.10.1 的回复：无法访问主机
请求超时
请求超时
请求超时

10.10.10.1 的 Ping 统计信息:
    数据包: 已发送 = 4，已接收 = 1，丢失 = 3 (75% 丢失)，
```

至此基于 Web UI 的业务发放实验完成。

4.5 使用 Python 实现业务发放

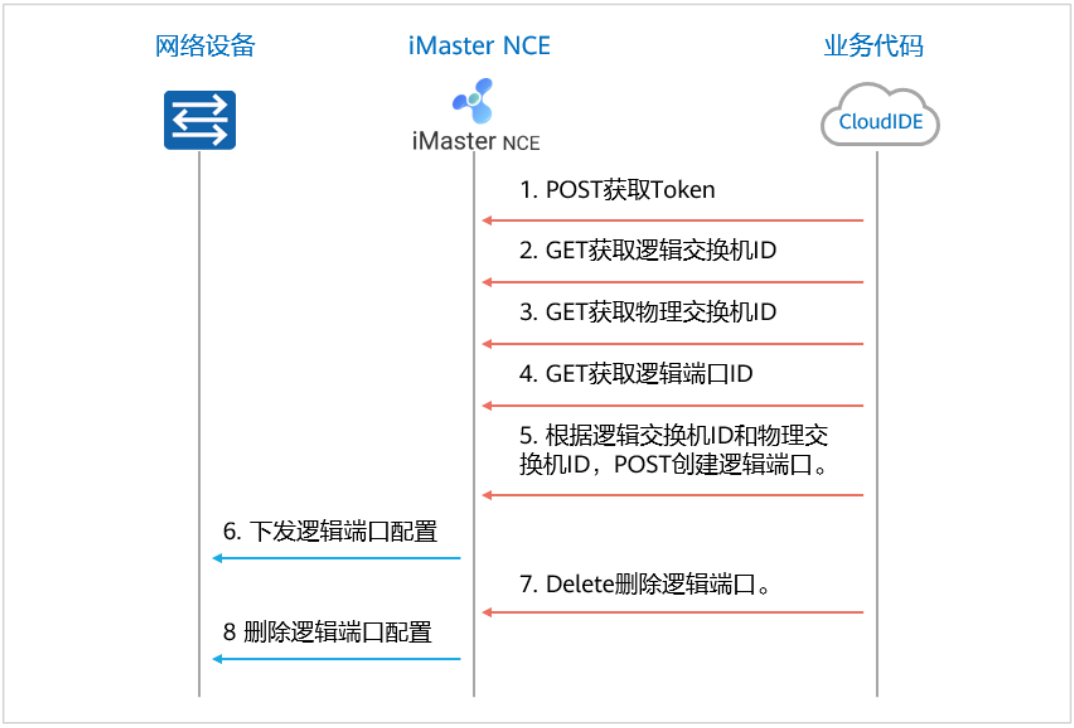
本小节您将尝试编写一个业务发放 Demo，调用 iMaster NCE-Fabric 北向开放 API 实现逻辑端口的创建与删除。

iMaster NCE-Fabric 上已在租户“Tenant_01”中创建好 VPC“VPC01”、逻辑路由器“Router”、逻辑交换机“ls2”和逻辑端口“LP_Office”。可点击查看其逻辑网络如下：



现在需要在逻辑交换机“ls2”下创建一个新的逻辑端口“LP_Office_Access”与“LP_Office”二层互通。逻辑端口的对应的物理交换机名称为“GW2”（可以在“网络规建”>“物理资源”>“设备管理”>“设备”中查看）。

本实验各组件具体交互流程如下：



业务发放代码需实现六项功能：

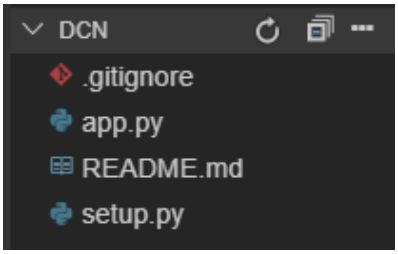
- 使用北向账户获取 token；
- 获取逻辑交换机 ID；

- 获取物理设备 ID；
- 获取逻辑端口 ID；
- 创建逻辑端口；
- 删除逻辑端口；

4.5.1 代码目录结构

本 Demo 基于 Python 编写。

代码结构如下：



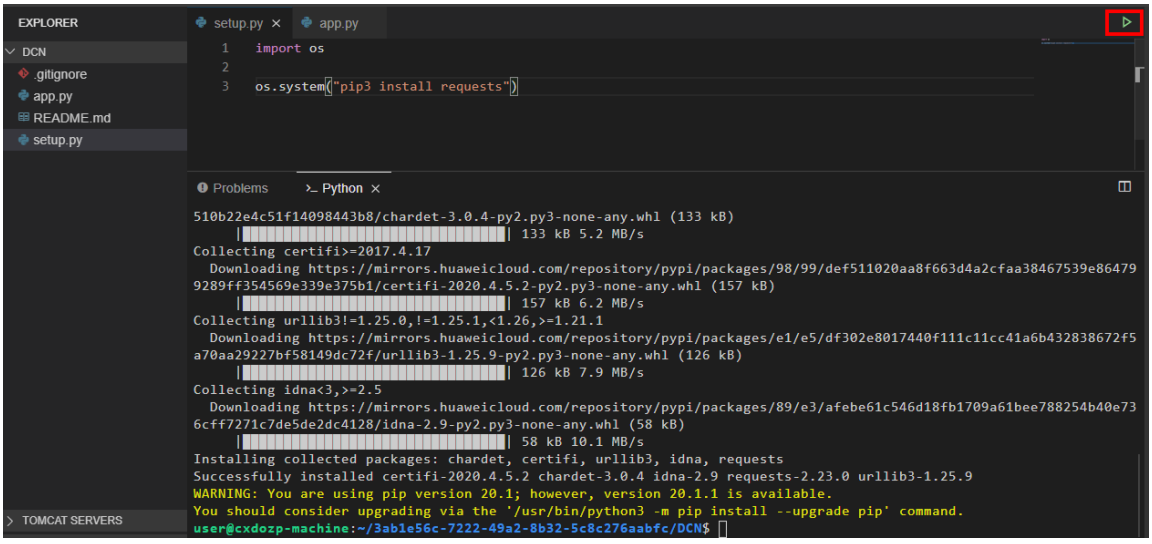
- app.py 为本应用的主要业务逻辑。
- setup.py 作为安装应用的依赖包。

4.5.2 安装 requests

setup.py 包含安装代码。

```
import os

os.system("pip3 install requests")os.system("pip3 install --user -r {}".format(requirements))
```

运行 setup.py 代码安装 requests 模块。



安装成功。

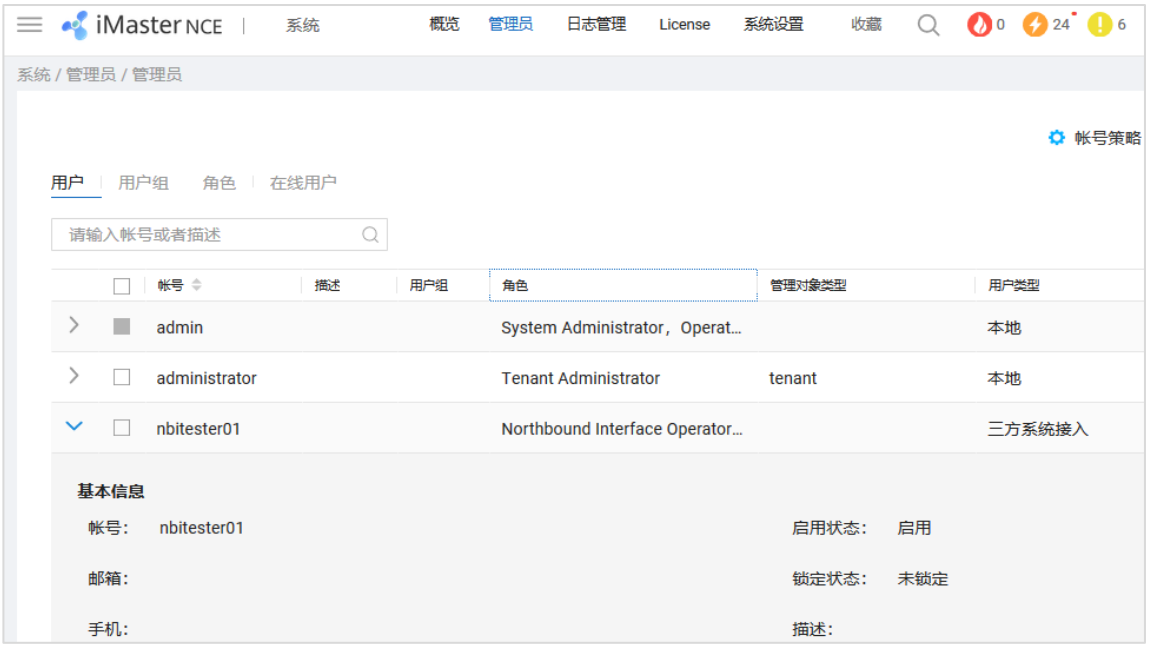
4.5.3 业务代码主函数

主函数呈现整体业务流程。

```
if __name__ == "__main__":
    # 配置北向用户信息及北向地址
    nbi_name = "nbitester01"
    nbi_pwd = "Huawei@123"
    host = "218.2.129.55"
    port = "18002"

    token_id = get_token_id(nbi_name, nbi_pwd, host, port)
    switch_id = get_switch_id(host, port, token_id, "ls2")
    device_id = get_device_id(host, port, token_id, "GW2")
    port_id = get_logicport_id(host,port,token_id,"LP_OfficeAccess")
    if port_id == "":
        create_logicport_id(host, port, token_id, "LP_OfficeAccess",switch_id, device_id)
    else:
        del_logicport_id(host,port,token_id,port_id)
```

首先配置北向用户和其登录信息。

北向账户可在“系统”>“管理员”>“管理员”页面查看或创建。



定义六个函数，其作用为：

- get_token_id(nbi_name, nbi_pwd, host, port)，输入北向账户信息获得 Token id。
- get_switch_id(host, port, token_id, "ls2")，获取名为“ls2”的逻辑交换机 ID。
- get_device_id(host, port, token_id, "GW2")，获取名为“GW2”的设备 ID。
- get_logicport_id(host,port,token_id,"LP_OfficeAccess")，获取名为“LP_OfficeAccess”的逻辑端口 ID。

- create_logicport_id(host, port, token_id, "LP_OfficeAccess",switch_id, device_id)，在指定的 switch_id 和 device_id 下创建名为“LP_OfficeAccess”的逻辑端口。
- del_logicport_id(host,port,token_id,port_id)，删除指定 port_id 的逻辑端口。

4.5.4 获取 Token ID

将获取 Token ID 的功能封装为函数 get_token_id。

```
def get_token_id(nbi_name, nbi_pwd, host, port):
    # 配置调用 token 的 url
    POST_TOKEN_URL = "/controller/v2/tokens"
    # 配置 URL 和 Headers
    post_token_url = "https://" + host + ":" + port + POST_TOKEN_URL
    headers_post = {'Content-Type': 'application/json', 'Accept': 'application/json'}
    # 发起请求，添加 Json 格式数据
    r = requests.post(post_token_url, headers=headers_post, json={"userName": nbi_name, "password": nbi_pwd},
verify=False)
    # 解析 token_id
    token_id = r.json()['data']['token_id']
    print("1.【Get Token Id】")
    print("【post_token_url】: "+post_token_url)
    print("【token_id】: "+token_id)
    return token_id
```

获取 Token 对应的 URL 和 HTTP 请求消息请参考版本匹配的《iMaster NCE-Fabric 产品文档》“二次开发”>“接口参考”>“用户认证”>“获取 token”。

最后从 HTTP 响应消息中解析出 Token ID 并返回。

4.5.5 获取逻辑交换机 ID

将获取逻辑交换机 ID 功能封装为函数。

```
def get_switch_id(host,port,token_id, switch_name):
    switch_id = ""
    GET_LOGICSWITCH_URL = "/controller/dc/v3/logicnetwork/switchs"
    # 配置 URL 和 Headers
    get_switchs_url = "https://" + host + ":" + port + GET_LOGICSWITCH_URL
    headers_get = {'Content-Type': 'application/json', 'Accept': 'application/json', 'X-AUTH-TOKEN':token_id}
    #调用获取 switch 接口
    r = requests.get(get_switchs_url, headers=headers_get, verify=False)
    print("2.【Get Switch Info】")
    print("【get_Switch_url】: "+get_switchs_url)
    switchs_info = r.json()['switch']
    for singleswitch    in switchs_info:
        if singleswitch.get("name") == switch_name:
            switch_id= singleswitch.get("id")
            break
    print("【Switch_id】: "+str(switch_id))
```

```
    return switch_id
```

查看产品文档“二次开发”>“接口参数”>“VPC”>“逻辑交换机”>“查询所有逻辑交换机信息”。其对应的调用方法为“GET”，URI 为“/controller/dc/v3/logicnetwork/switchs”。

逻辑交换机的 HTTP 请求中可以包含可选字段，本例中仅有必选字段。

iMaster NCE-Fabric 的相应报文中包含“switch”字段携带逻辑交换机信息，完整信息为：

{'switch': [{'id': '28d63a1a-2e68-43fe-9644-942e4266e52c', 'name': 'Switch1', 'description': '', 'logicNetworkId': '74173b4e-cb4b-461d-8117-cacdfa70430f', 'vni': 5000, 'bd': None, 'macAddress': '00:00:5E:00:01:02', 'tenantId': 'b37e91e1-b8f7-45c0-a03e-6c6968245530', 'additional': {'producer': 'component-ui', 'createAt': '2020-06-08 20:08:43', 'updateAt': '2020-06-08 20:08:44'}, 'stormSuppress': {'broadcastEnable': False, 'multicastEnable': False, 'unicastEnable': False, 'broadcastCbs': None, 'broadcastCbsUnit': None, 'broadcastCir': None, 'broadcastCirUnit': None, 'unicastCbs': None, 'unicastCbsUnit': None, 'unicastCir': None, 'unicastCirUnit': None, 'multicastCbs': None, 'multicastCbsUnit': None, 'multicastCir': None, 'multicastCirUnit': None}, 'importRouteTargets': [], 'exportRouteTargets': [], 'qosTemplateId': ''}, {'id': 'caae22d0-dfb6-44b2-896f-593c942affac', 'name': 'ls2', 'description': '', 'logicNetworkId': 'aed8487a-6876-4bea-8b11-f8551aa1b878', 'vni': 5006, 'bd': 5001, 'macAddress': '00:00:5E:00:01:02', 'tenantId': 'ce8bfbd3-5dac-421c-b97a-bb8dd35f5b37', 'additional': {'producer': 'component-ui', 'createAt': '2020-05-25 16:22:39', 'updateAt': '2020-05-27 15:54:29'}, 'stormSuppress': {'broadcastEnable': False, 'multicastEnable': False, 'unicastEnable': False, 'broadcastCbs': None, 'broadcastCbsUnit': None, 'broadcastCir': None, 'broadcastCirUnit': None, 'unicastCbs': None, 'unicastCbsUnit': None, 'unicastCir': None, 'unicastCirUnit': None, 'multicastCbs': None, 'multicastCbsUnit': None, 'multicastCir': None, 'multicastCirUnit': None}, 'importRouteTargets': [], 'exportRouteTargets': [], 'qosTemplateId': ''}, {'id': '10000002-0000-0000-0002-000000000003', 'name': 'LS_Office', 'description': 'LS_Office', 'logicNetworkId': 'aed8487a-6876-4bea-8b11-f8551aa1b878', 'vni': 6002, 'bd': 6002, 'macAddress': '00:00:5E:00:01:02', 'tenantId': 'ce8bfbd3-5dac-421c-b97a-bb8dd35f5b37', 'additional': {'producer': 'default', 'createAt': '2020-05-29 16:08:46', 'updateAt': None}, 'stormSuppress': {'broadcastEnable': False, 'multicastEnable': False, 'unicastEnable': False, 'broadcastCbs': None, 'broadcastCbsUnit': None, 'broadcastCir': None, 'broadcastCirUnit': None, 'unicastCbs': None, 'unicastCbsUnit': None, 'unicastCir': None, 'unicastCirUnit': None, 'multicastCbs': None, 'multicastCbsUnit': None, 'multicastCir': None, 'multicastCirUnit': None}, 'importRouteTargets': [], 'exportRouteTargets': [], 'qosTemplateId': None}], 'totalNum': 3, 'pageIndex': 1, 'pageSize': 3}

这里我们只需要获取指定 switch_name 的 id。所以使用 for 循环匹配后返回。

4.5.6 获取物理设备 ID

将获取物理设备 ID 功能封装为函数。

```
def get_device_id(host,port,token_id, device_name):
    device_id = ""
    GET_DEVICE_URL = "/acdcn/v3/topoapi/dcntopo/device"
    #  配置 URL 和 Headers
    get_devices_url = "https://" + host + ":" + port + GET_DEVICE_URL
    headers_get = {'Content-Type': 'application/json', 'Accept': 'application/json', 'X-AUTH-TOKEN':token_id}
    #调用获取 device 接口
    r = requests.get(get_devices_url, headers=headers_get, verify=False)
    print("3.【Get Device Info】")
    print("【get_Device_url】: "+get_devices_url)
    devices_info = r.json()['devices']
    for singlesdevice    in devices_info:
        if singlesdevice.get("name") == device_name:
            device_id= singlesdevice.get("id")
            break
    print("【Device_id】: "+str(device_id))
```

```
    return device_id
```

同样查阅二次开发文档获取对应的 URI 和方法。发起 HTTP 请求。

在 HTTP 相应中提取需要查询的 device_name 对应的 id，返回。

4.5.7 获取逻辑端口 ID

将获取逻辑端口 ID 封装为函数。

```
def get_logicport_id(host,port,token_id, port_name):
    port_id = ""
    GET_PORT_URL = "/controller/dc/v3/logicnetwork/ports"
    #  配置 URL 和 Headers
    get_ports_url = "https://" + host + ":" + port + GET_PORT_URL
    headers_get = {'Content-Type': 'application/json', 'Accept': 'application/json', 'X-AUTH-TOKEN':token_id}
    #调用获取逻辑端口接口
    r = requests.get(get_ports_url, headers=headers_get, verify=False)
    print("4.【Get Port Info】")
    print("【get_Port_url】: "+get_ports_url)
    devices_info = r.json()['port']
    for singlesdevice    in devices_info:
        if singlesdevice.get("name") == port_name:
            port_id= singlesdevice.get("id")
            break
    print("【Port_id】: "+str(port_id))
    return port_id
```

查询二次开发文档获取查询逻辑端口的方法和 URI。

使用 request.get 发起 HTTP 请求。返回匹配 port_name 的 port_id。

4.5.8 创建逻辑端口

将创建逻辑端口功能封装为函数。

```
def create_logicport_id(host, port, token_id, port_name,switch_id, device_id):
    CREATE_PORT_URL = "/controller/dc/v3/logicnetwork/ports"
    port_id = "10000002-0000-0000-0000-000000000199"
    port_info = "10GE1/0/15"
    #  配置 URL 和 Headers
    create_ports_url = "https://" + host + ":" + port + CREATE_PORT_URL
    headers_get = {'Content-Type': 'application/json', 'Accept': 'application/json', 'X-AUTH-TOKEN':token_id}
    bodydata =
{"port":[{"logicSwitchId":switch_id,"name":port_name,"description":port_name,"id":port_id,"accessInfo":{"mode":"
UNI","location":[{"portName":port_info,"deviceId":device_id}],"type":"UNTAG"}}]}
    #调用创建逻辑端口接口
    r = requests.post(create_ports_url, headers=headers_get, data=json.dumps(bodydata),verify=False)
    print("5.【Create Port Info】")
    print("【Create_Port_url】: "+create_ports_url)
    if r.status_code == 204:
        print("【Create_Port_successfully】")
```

```
    else:
        print("【Create_Port_unsuccessfully】")
```

查询产品文档“二次开发”，获取创建逻辑端口的方法和 URI。

注意查阅手册此处创建逻辑端口的 HTTP 请求有必须携带的参数“port”。“port”中必须携带的参数有“id”、“name”、“logicSwitchId”、“accessInfo”四个参数，“description”为可选参数。

其中“accessInfo”中“mode”、“type”和“location”为必选参数。

- “location”中携带“portName”，“deviceId”参数，标示逻辑端口名称和对应配置下发的物理设备。
- “type”字段表示逻辑端口类型。因为测试终端发出流量不携带 VLAN 标签，所以此处为 UNTAG。
- “mode”表示端口模式，和基于 Web UI 发放过程一致为 UNI。

根据以上信息构建 HTTP 请求，如果返回状态码为 204 则表示创建成功。

4.5.9 删除逻辑端口

将删除逻辑端口功能封装为函数。

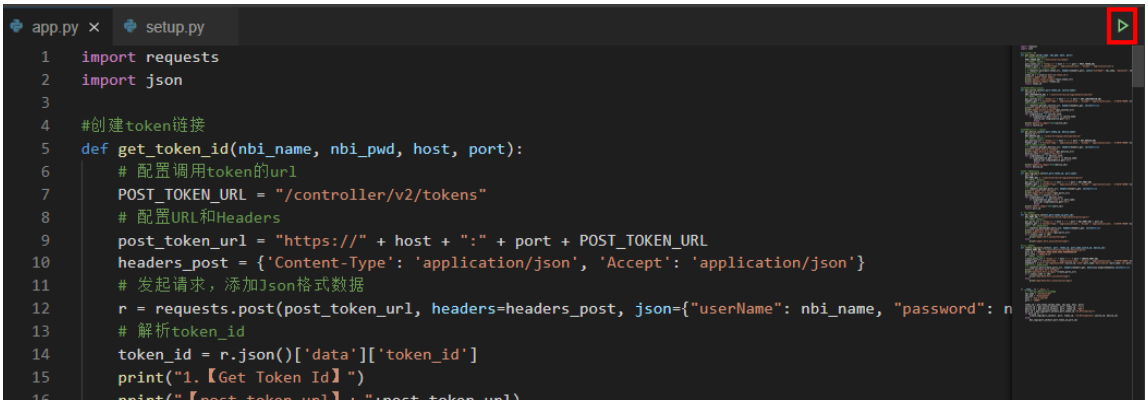
```
def del_logicport_id(host,port,token_id,port_id):
    DEL_PORT_URL = "/controller/dc/v3/logicnetwork/ports/port/"
    #  配置 URL 和 Headers
    get_ports_url = "https://" + host + ":" + port + DEL_PORT_URL + port_id
    headers_get = {'Content-Type': 'application/json', 'Accept': 'application/json', 'X-AUTH-TOKEN':token_id}
    #调用删除逻辑端口接口
    r = requests.delete(get_ports_url, headers=headers_get, verify=False)
    print("5.【Delete Port Info】")
    print("【del_Port_url】: "+get_ports_url)
    if r.status_code == 204:
        print("【del_Port_successfully】")
    else:
        print("【del_Port_unsuccessfully】")
```

查询产品文档“二次开发”，获取创建逻辑端口的方法为 DELETE 和 URI 为 /controller/dc/v3/logicnetwork/ports/port/{id}。

根据二次开发文档构建 HTTP 请求，如果返回码为 204 表示逻辑端口删除成功。

4.5.10 结果验证

1.CloudIDE 中运行代码，创建逻辑端口。



在交互界面的输出如下：

user@hbbsjw-machine:~/915f9564-fbeb-4638-99e5-c8de6547e82b/DCN$ /usr/bin/python /home/user/915f9564-fbeb-4638-99e5-c8de6547e82b/DCN/app.py
/home/user/.local/lib/python3.6/site-packages/urllib3/connectionpool.py:986: InsecureRequestWarning: Unverified HTTPS request is being made to host '218.2.129.55'. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
InsecureRequestWarning,
1.【Get Token Id】

【post_token_url】：https://218.2.129.55:18002/controller/v2/tokens

【token_id】：x-88leek7t49jzikpe1duo5gc6fv6pikmk1dur3tbsmotj3vc95fepqorx48eppfo989041hby6pjyvuobnshis8dgfute9eapanpj455e9j8588sanw1e7uqmc6buqr1f
/home/user/.local/lib/python3.6/site-packages/urllib3/connectionpool.py:986: InsecureRequestWarning: Unverified HTTPS request is being made to host '218.2.129.55'. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
InsecureRequestWarning,
2.【Get Switch Info】

【get_Switch_url】：https://218.2.129.55:18002/controller/dc/v3/logicnetwork/switchs

【Switch_id】：caae22d0-dfb6-44b2-896f-593c942affac

/home/user/.local/lib/python3.6/site-packages/urllib3/connectionpool.py:986: InsecureRequestWarning: Unverified HTTPS request is being made to host '218.2.129.55'. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
InsecureRequestWarning,
3.【Get Device Info】

【get_Device_url】：https://218.2.129.55:18002/acdcn/v3/topoapi/dcntopo/device

【Device_id】：55e8bee3-73de-3041-b266-103e0314df14

/home/user/.local/lib/python3.6/site-packages/urllib3/connectionpool.py:986: InsecureRequestWarning: Unverified HTTPS request is being made to host '218.2.129.55'. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
InsecureRequestWarning,
4.【Get Port Info】

【get_Port_url】：https://218.2.129.55:18002/controller/dc/v3/logicnetwork/ports

【Port_id】：

/home/user/.local/lib/python3.6/site-packages/urllib3/connectionpool.py:986: InsecureRequestWarning: Unverified HTTPS request is being made to host '218.2.129.55'. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
InsecureRequestWarning,
5.【Create Port Info】

【Create_Port_url】：https://218.2.129.55:18002/controller/dc/v3/logicnetwork/ports
【Create_Port_successfully】

可以看到业务代码成功与 iMaster NCE-Fabric 交互，最后成功创建端口。

2.登录 iMaster NCE-Fabric 查看

在“业务发放”>“VPC”中，查看 vpc_01 的逻辑网络情况。

可以看到已成功创建名为“LP_OfficeAccess”的逻辑端口。

3.CloudIDE 再次运行代码，删除逻辑端口。

再次运行代码。

输出如下：

```
1.【Get Token Id】
【post_token_url】：https://218.2.129.55:18002/controller/v2/tokens
【token_id】：x-
5himjuul2oo9868bqmk5s7bynu9eqoc51ec4fyjxcarsk7mrs6epdgjx9crzkb3u7yntvwvxk72kpg7xtheqbv7y2murel2pbt6k
4ak6peemthqnk6o6ledcfzfys8er
/home/user/.local/lib/python3.6/site-packages/urllib3/connectionpool.py:986: InsecureRequestWarning: Unverified
HTTPS request is being made to host '218.2.129.55'. Adding certificate verification is strongly advised. See:
https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
  InsecureRequestWarning,
2.【Get Switch Info】
【get_Switch_url】：https://218.2.129.55:18002/controller/dc/v3/logicnetwork/switchs
【Switch_id】：caae22d0-dfb6-44b2-896f-593c942affac
/home/user/.local/lib/python3.6/site-packages/urllib3/connectionpool.py:986: InsecureRequestWarning: Unverified
HTTPS request is being made to host '218.2.129.55'. Adding certificate verification is strongly advised. See:
https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
  InsecureRequestWarning,
3.【Get Device Info】
【get_Device_url】：https://218.2.129.55:18002/acdcn/v3/topoapi/dcntopo/device
【Device_id】：55e8bee3-73de-3041-b266-103e0314df14
/home/user/.local/lib/python3.6/site-packages/urllib3/connectionpool.py:986: InsecureRequestWarning: Unverified
HTTPS request is being made to host '218.2.129.55'. Adding certificate verification is strongly advised. See:
https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
  InsecureRequestWarning,
4.【Get Port Info】
【get_Port_url】：https://218.2.129.55:18002/controller/dc/v3/logicnetwork/ports
【Port_id】：10000002-0000-0000-0000-000000000199
/home/user/.local/lib/python3.6/site-packages/urllib3/connectionpool.py:986: InsecureRequestWarning: Unverified
HTTPS request is being made to host '218.2.129.55'. Adding certificate verification is strongly advised. See:
https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
  InsecureRequestWarning,
5.【Delete Port Info】
【del_Port_url】：https://218.2.129.55:18002/controller/dc/v3/logicnetwork/ports/port/10000002-0000-0000-0000-
000000000199
【del_Port_successfully】
```

第二次运行代码步骤 5 日志变为【Delete Port Info】，成功删除了逻辑端口。

4.登录 iMaster NCE-Fabric 查看

在“业务发放”>“VPC”中，查看 vpc_01 的逻辑网络情况。

名为“LP_OfficeAccess”的逻辑端口被删除，拓扑恢复原状。

至此使用 Python 实现业务发放实验结束。

4.6 思考题

1.本例中如果存在重名的逻辑交换机，如何获取其逻辑交换机 ID？

2.如何通过 Python 代码实现 VPC、逻辑路由器和逻辑交换机的创建？

思考题参考答案

《RESTful API 调用实践》参考答案：

本实验中 Python 代码和 iMaster NCE-Campus 有两次交互，首先使用发起 POST 请求获得 Token，然后发起 GET 请求查询站点信息。两次请求的方法、URI 和 body 都不同。HTTP 请求消息需要遵守“RESTful API 开发指南”要求。

《智简园区无线定位实践》参考答案：

本实验中获取到的数据可以根据实际项目需求进一步处理，例如写入对应的数据库或者推送到消息队列等。

《智简数据中心业务发放实践》参考答案：

1.本实验如果出现重名的逻辑交换机，可以通过匹配租户或 VPC 等更多字段指定特定对象。

2.在产品文档中“二次开发”获取 VPC、逻辑路由器和逻辑交换机相关信息，然后根据前后顺序完成创建。

华为认证系列教程

HCIP-Datacom

NCE业务开放可编程

实验指导手册

版本:1.0

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： http://e.huawei.com

华为认证体系介绍

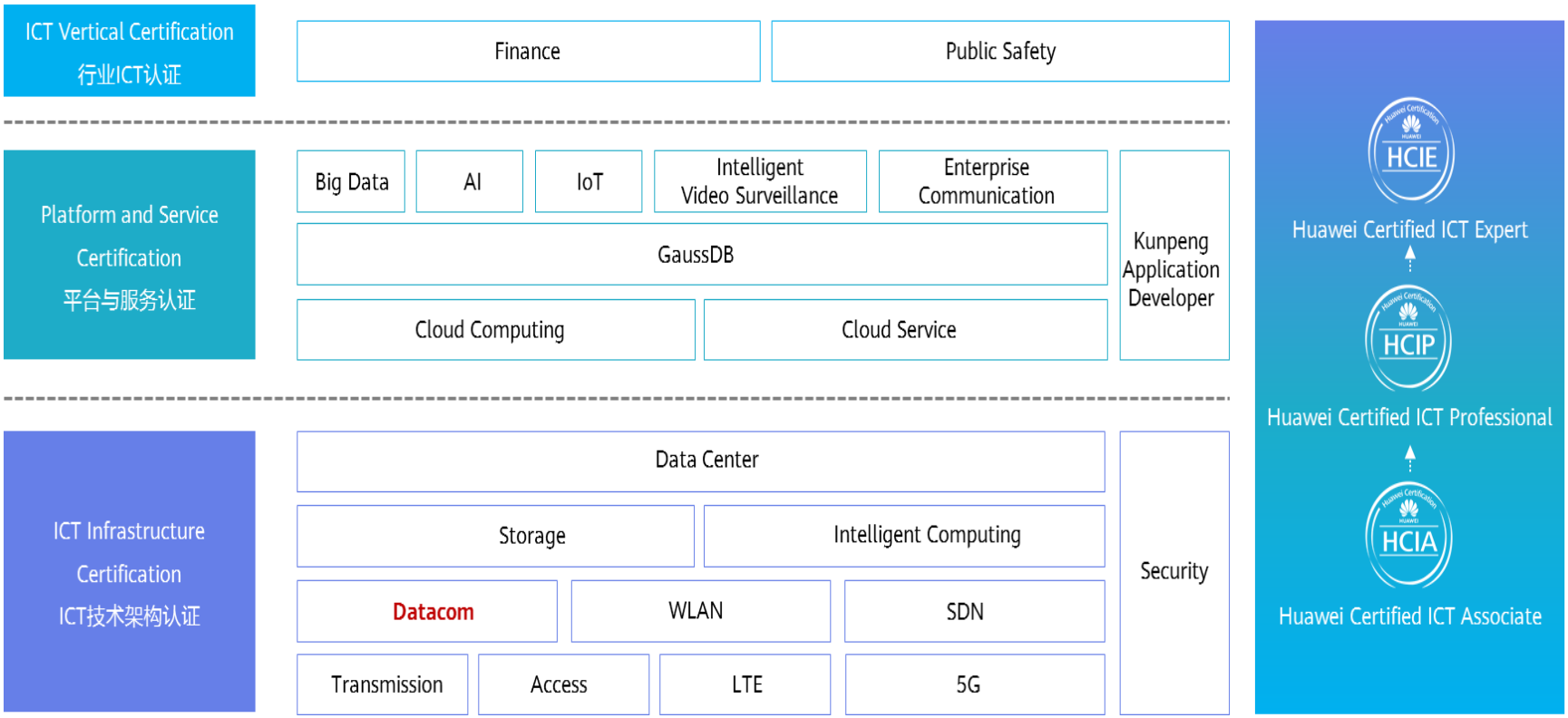
华为认证是华为公司基于“平台+生态”战略，围绕“云-管-端”协同的新ICT技术架构，打造的ICT技术架构认证、平台与服务认证、行业ICT认证三类认证，是业界唯一覆盖ICT（Information and Communications Technology 信息通信技术）全技术领域的认证体系。

根据ICT从业者的学习和进阶需求，华为认证分为工程师级别、高级工程师级别和专家级别三个认证等级。华为认证覆盖ICT全领域，符合ICT融合的技术趋势，致力于提供领先的人才培养体系和认证标准，培养数字化时代新型ICT人才，构建良性ICT人才生态

HCIP-Datacom-Network Automation Developer定位于培养数通网络领域具备网络自动化开发专业知识和技能水平的高级工程师。通过HCIP-Datacom-Network Automation Developer认证将证明您能够胜任企业网络自动化开发工程师岗位，具备使用华为数通设备进行企业网络自动化部署、开发和运维的能力。

华为认证协助您打开行业之窗，开启改变之门，屹立在数通领域的潮头浪尖！

Huawei Certification



前 言

简介

本书为 HCIP-Datacom-Network Automation Developer 认证培训教程，适用于准备参加 HCIP-Datacom-Network Automation Developer 考试的学员，或者希望了解华为 iMaster NCE 业务开放可编程基础知识和实践的读者。

读者知识背景

本文档主要适用于进阶学习的网络自动化工程师。读者需具备以下知识和技能：

- Python 编程基础
- 华为云代码托管
- RESTful 原理
- NETCONF YANG 原理
- Jinja2 模板
- 了解华为 iMaster NCE 业务开放可编程

实验环境说明

环境介绍

本实验环境基于智简网络开发者社区。

智简网络开发者社区是面向数据通信领域依托华为公有云 DevCloud 开发服务，提供开发者和合作伙伴的“学习、开发、测试、交流”一站式服务平台。目前提供华为数通网络开放可编程，智简园区网络，智简数据中心网络，广域网络等解决方案的开放场景，以及 API Explorer、API Studio、沙箱、DevOps 开发 IDE 与 SDK 等开发资源，帮助开发者快速体验、开发、集成和上线各行业应用。

本手册将介绍使用开发者社区环境 iMaster NCE 进行业务开放可编程实践。

使用说明

- 注册华为云帐号并实名认证：https://www.huaweicloud.com/。
- 访问智简网络开发者社区：https://developer.huaweicloud.com/resource/network.html，选择“数通网络开放可编程”获取更多信息。

本地编译环境准备

安装 Python3、Pycharm、Gpg4Win、业务开放可编程 SDK。

- 安装 Python3（具体步骤查看《HCIP-Datacom-Python 编程基础实验手册》）
- 安装 Pycharm（略）
- 安装 Gpb4Win

官网下载 Gpg4Win 安装包（https://www.gpg4win.org/），根据向导，选择默认的配置，单击安装直到完成。

- 安装业务开放可编程 SDK

从开发者社区“数通网络开放可编程”>“资源下载”下载 SDK 到本地。

输入 CMD 命令打开“命令提示符”窗口，将地址切换到 python-aoc-api-xxx.rar 文件解压的所在目录。输入 dir 命令，按回车键查看目录下的相关文件

```
D:\AOC\SDK>dir
2020/04/09    16:02               121,569 aoc_api-2.0.0-py3-none-any.whl
               1 个文件            121,569 字节
               2 个目录  103,275,847,680 可用字节
```

输入 pip install aoc_api-2.0.0-py3-none-any.whl 命令，按回车键安装 SDK 文件。

```
D:\AOC\SDK>pip install aoc_api-2.0.0-py3-none-any.whl
Looking in indexes: http://mirrors.tools.huawei.com/pypi/simple
Processing d:\download\sdk\aoc_api-2.0.0-py3-none-any.whl
….
Installing collected packages: netaddr, protobuf, aoc-api
Successfully installed aoc-api-2.0.0 netaddr-0.8.0 protobuf-3.12.2
```

安装成功。

目录

1 对接新设备-网元驱动包实践

1.1 实验背景

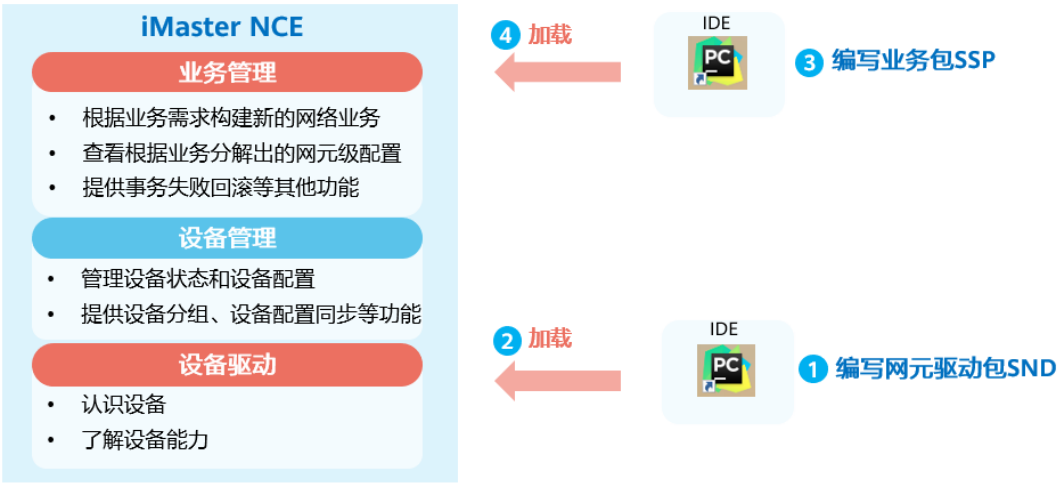
华为 iMaster NCE 业务开放可编程基于 YANG 模型驱动的开放架构，以网元驱动包和业务包的形式使能网元层和网络层开放可编程，自动生成配置页面和北向接口，实现新设备的快速对接和新网络业务的快速构建。

本实验将指导读者编写网元驱动包（SND，Specific NE Driver）开放设备原生能力，即使用 iMaster NCE 纳管网络设备并进行设备基础配置下发和北向接口的生成。本课程您会学习：

- 网元驱动包的编写过程
- 基于设备原生能力下发基础配置

1.1.1 网元驱动包介绍

网元驱动包（SND）是 iMaster NCE 软件包的一种，为开放可编程系统提供与网元交互的数据模型。该数据模型通常包含一个.py 文件和若干特性的数据模型（YANG）。前者用于定义网元的相关信息，如设备类型、厂商、连接信息等。后者描述了网元相关特性的数据结构。



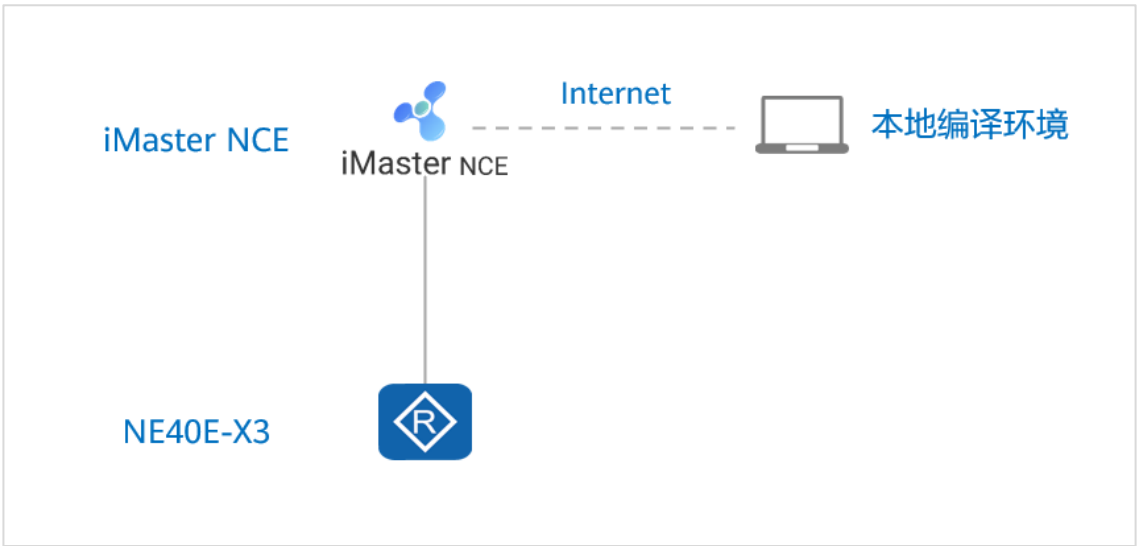
1.1.2 开放设备能力介绍

iMaster NCE 可以开放设备原生能力。基于设备 YANG 模型，NCE 业务开放可编程系统自动生成北向接口和配置页面，快速管理华为和三方设备。支持设备数据一致性的差异比较、配置对账、配置同步等功能。

本实验将主要介绍 iMaster NCE 设备能力开放流程和操作，更多能力介绍请学习 HCIP-Datacom-Network Automation Developer 教材《NCE 业务开放可编程》。

1.2 实验介绍

1.2.1 组网说明



本实验组网包含三个对象，NE40E-X3、iMaster NCE 和本地编译环境。其中 NE40E-X3、iMaster NCE 可由开发者社区提供沙箱环境，读者需在本地完成相关包的编译。

1.2.2 实验步骤

本实验步骤如下：

1. 环境准备：准备本地环境和实验沙箱环境。
2. 本地编写网元驱动包。
3. 使用 iMaster NCE 纳管设备。
4. 使用 iMaster NCE 下发设备基础配置（创建设备子接口为例）。
5. 使用北向接口下发设备配置。

1.3 环境准备

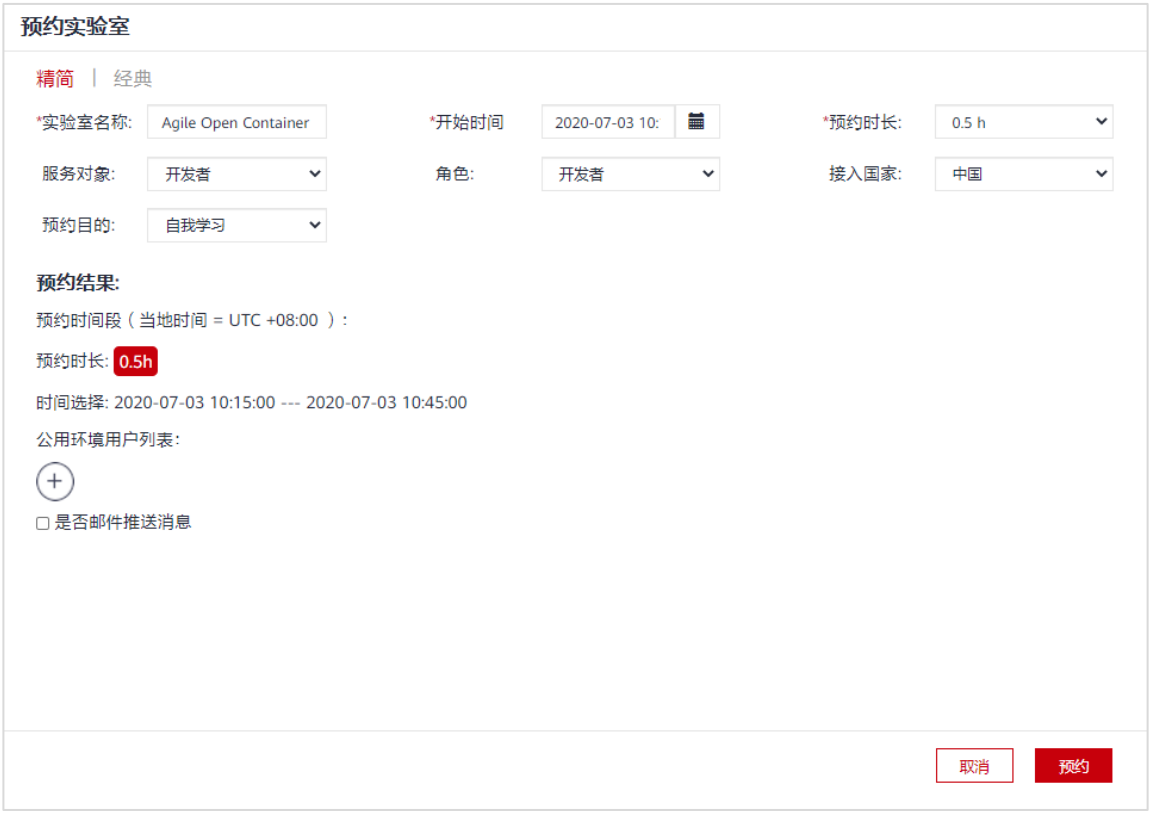
1.3.1 本地环境准备

本地环境准备参考“前言”>“实验环境说明”>“本地编译环境准备”。

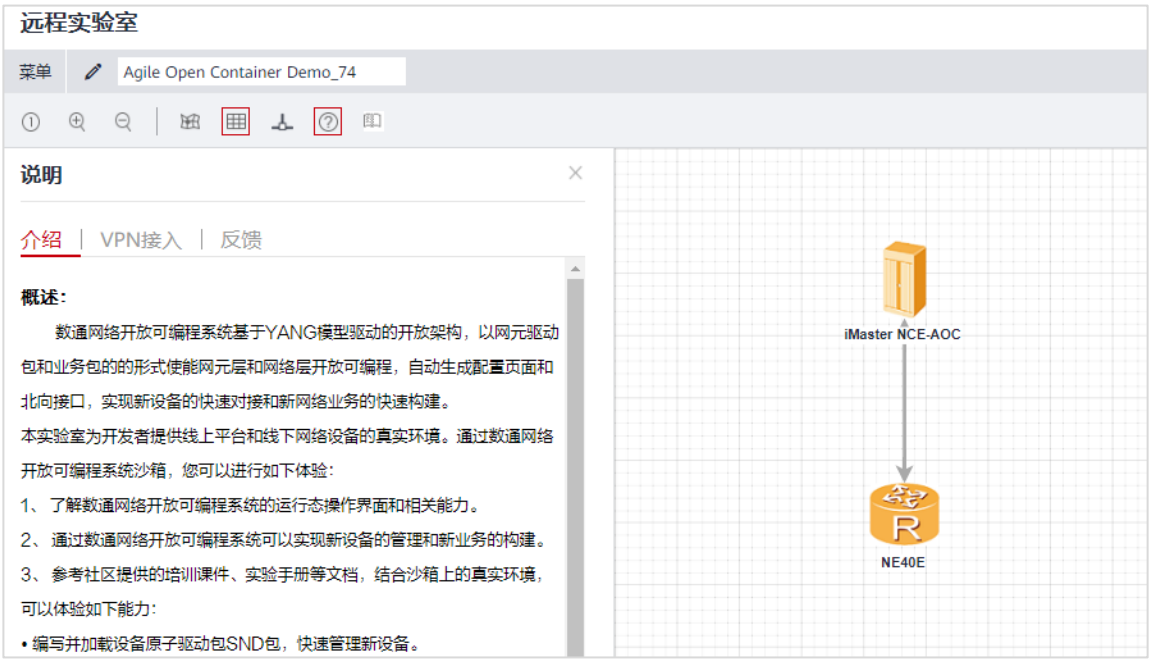
1.3.2 沙箱预约

开发者社区提供华为数通解决的沙箱体验，本实验申请在沙箱“远程实验室”中申请“Agile Open Container”，https://devzone.huawei.com/openecosystem/。

点击预约。



填写预约信息后即可进入沙箱环境。沙箱环境中包含拓扑和设备描述等详细信息。



1.4 编写网元驱动包（SND）

本章节您将尝试在本地环境编写 NE40E 的网元驱动包，并将其加载到 iMaster NCE。

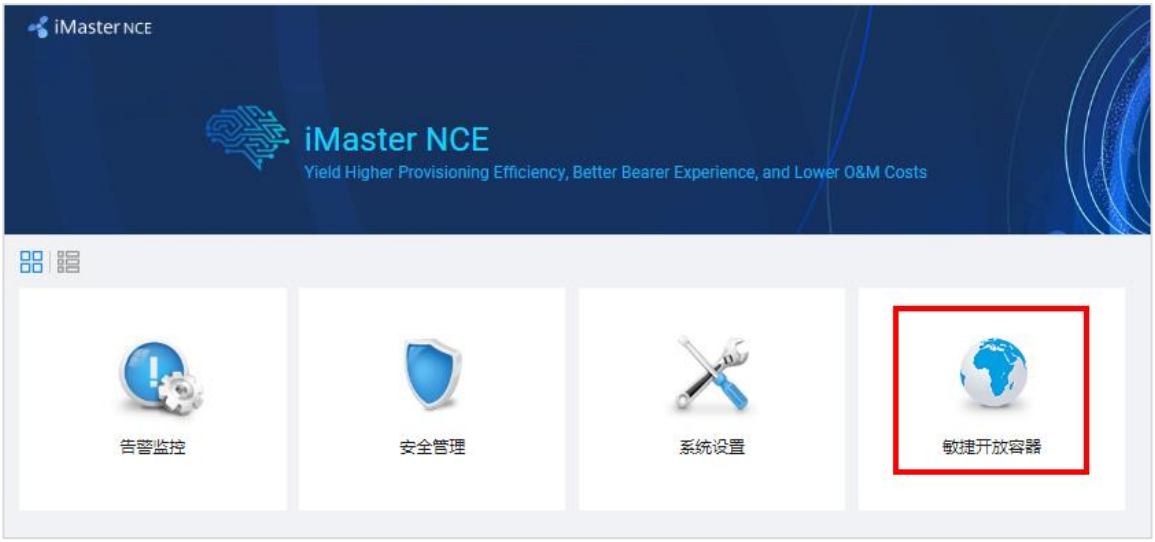
本实验的相关代码样例和资源文件可参考
https://devzone.huawei.com/apistudio/sample/aoc/apiSdk.html。

1.4.1 创建网元驱动包模板

1.登录 iMaster NCE。

沙箱环境中点击图中图标，根据“属性”信息登录 iMaster NCE。

首页中点击“敏捷开放容器”。



2.创建网元驱动包模板。

点击“工程管理”>“增加”，进入创建网元驱动包模板页面。

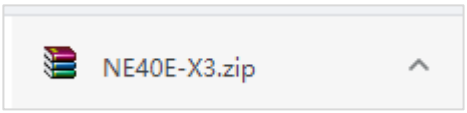
填写网元驱动包参数信息，填写完成之后单击“确定”按钮，完成模板创建。

- 名称：NE40E-X3
- 版本：1.0.0
- 提供商：HUAWEI
- 包类型：Specific NE driver
- 映射类型：Python
- 设备类型：NE40E-X3
- 协议类型：NETCONF

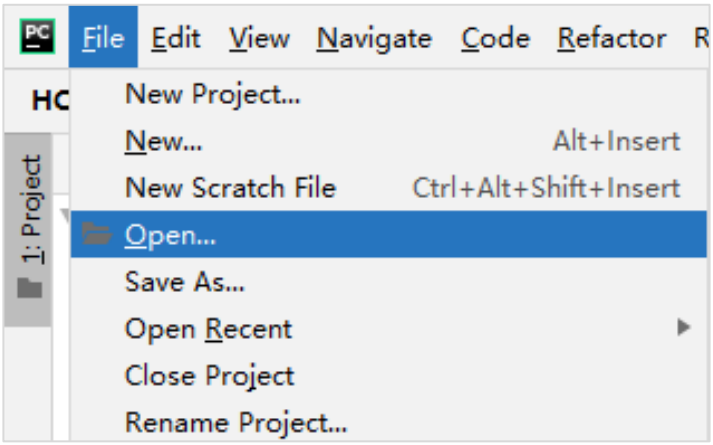
3.下载模板，导入本地 IDE Pycharm。

单击“操作”列的“导出”按钮，下载模板到本地。

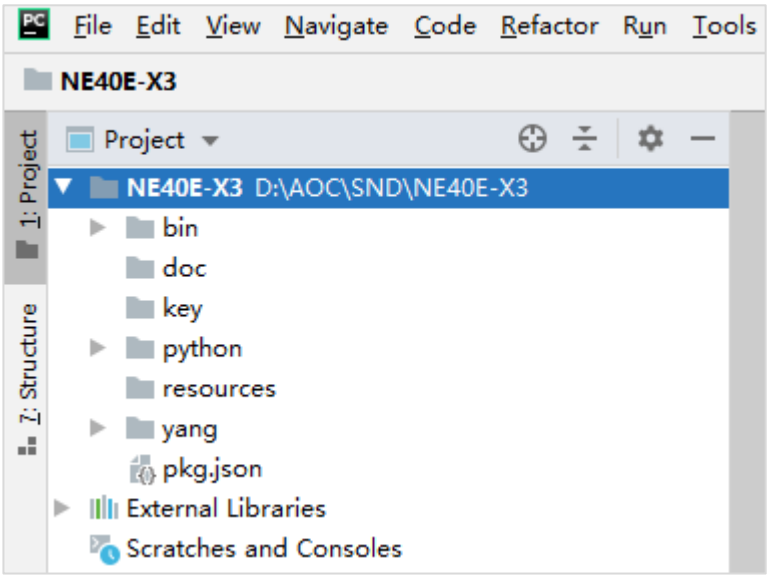
获得模板“NE40E-X3.zip”压缩文件。



将模板解压到指定本地目录，使用 Pycharm 打开此目录。



Pycharm 打开模板，其详细目录结构如下：



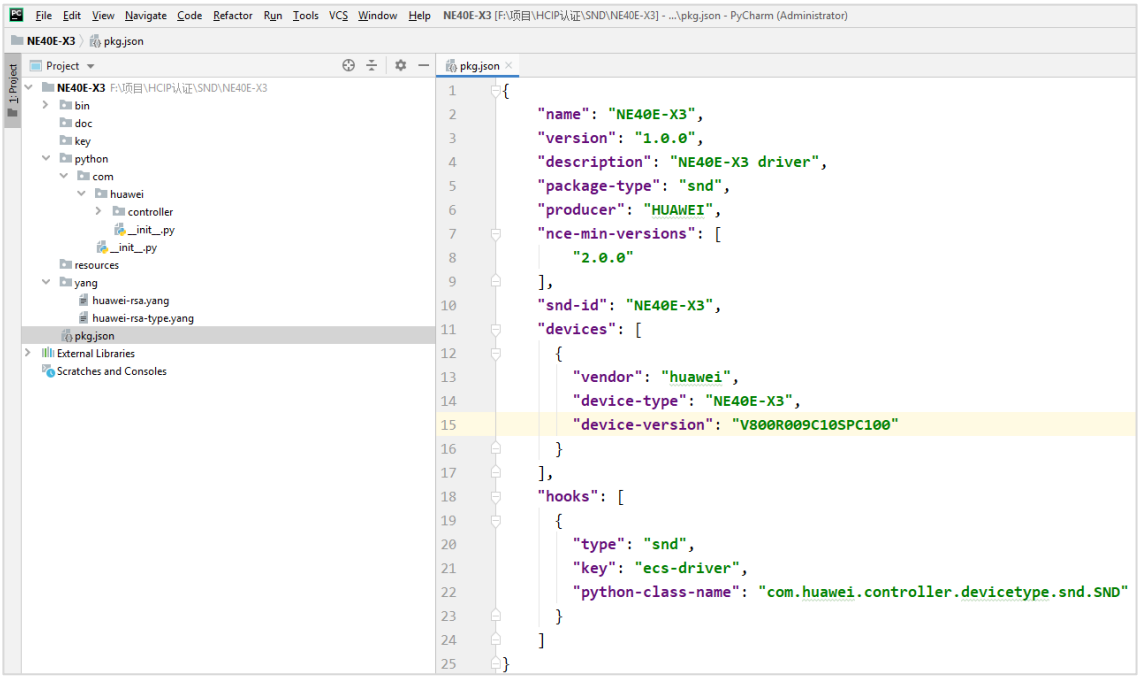
1. bin: 可执行脚本存放位置，包含打包的工具及脚本。
2. key: private 私钥存放位置。
3. python: Python 代码存放位置，包含用来实现业务回调逻辑的 python 脚本。

4. yang: 设备 YANG 模块。每个模块对应设备上的一个功能。它们共同组成设备 YANG 模型。

5. pkg.json: 包配置文件，用来设置当前软件包的基本属性和回调钩子。

1.4.2 修改包配置文件

修改包配置文件 pgk.json，修改为指定的设备参数。

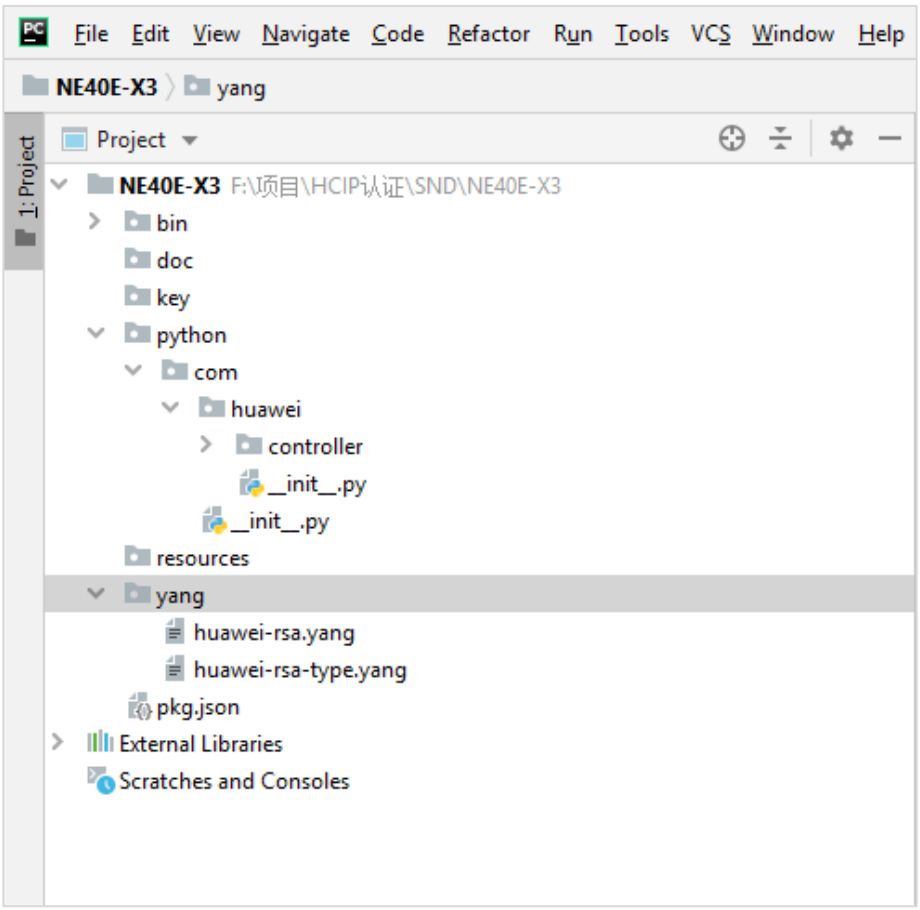


打开“pkg.json”，修改第 15 行，修改设备的版本号为正确的设备版本号，我们实验使用的是 V800R009C10SPC100。

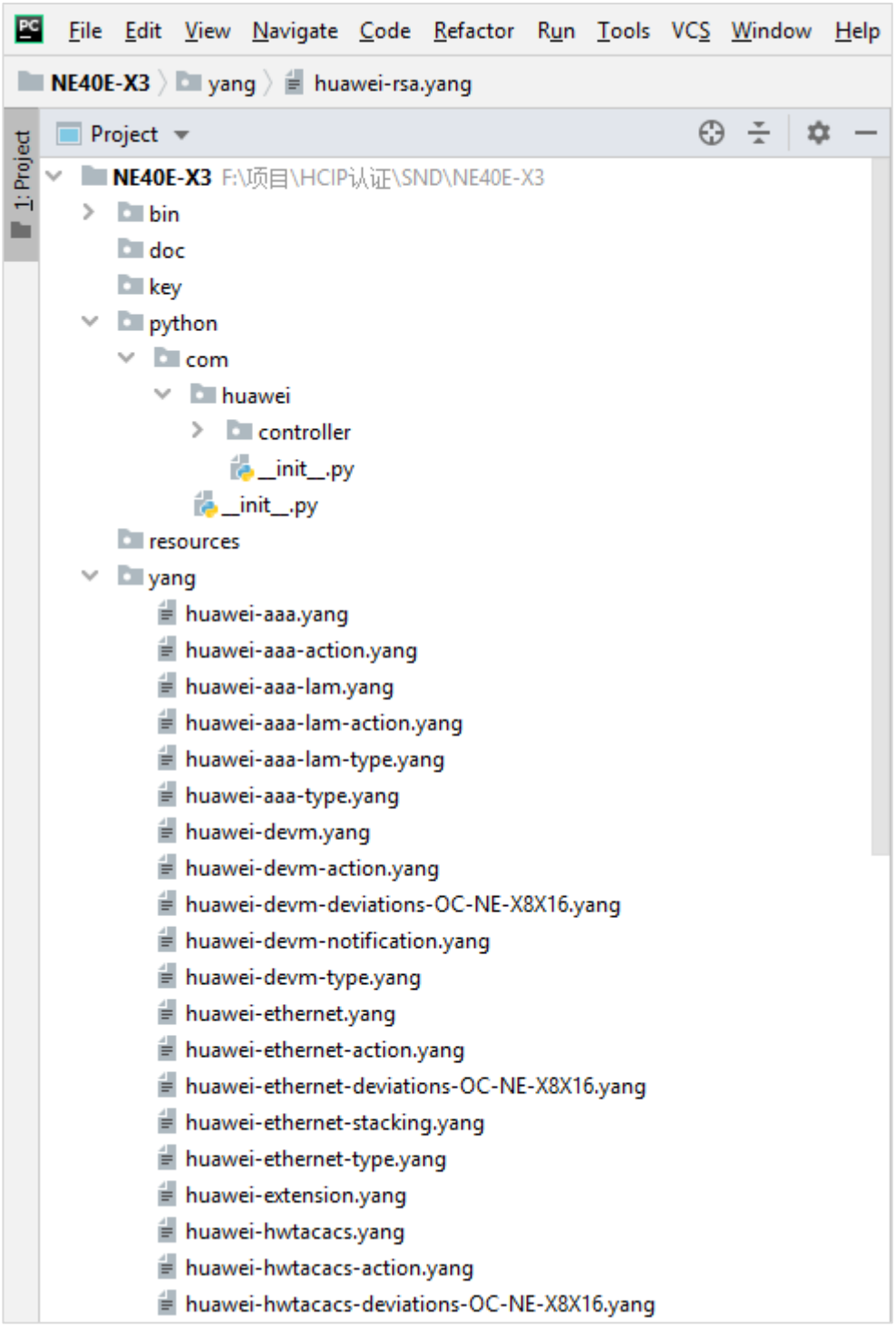
1.4.3 添加 YANG 文件

添加 NE40E-X3 对应版本的 YANG 文件到驱动包的 yang 目录。

首先将模板中的 yang 文件删除掉。



然后将 NE40E-X3 的 yang 文件（从设备厂商的网站上获取）复制到网元驱动包的 yang 目录下。



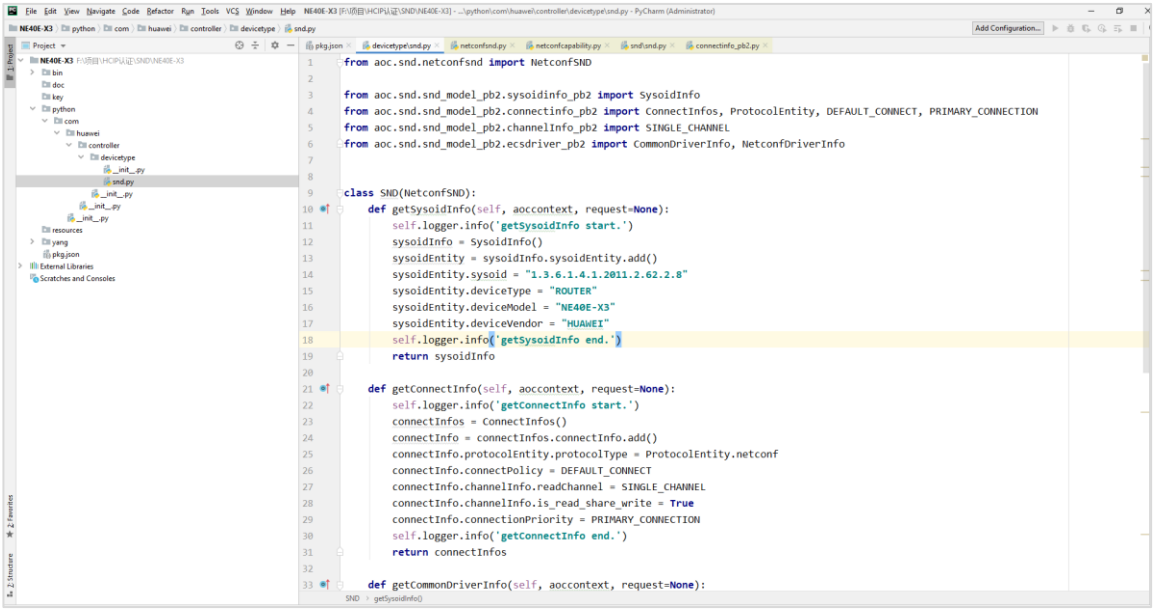
1.4.4 编写 Python 文件 snd.py

展开在网元驱动包的 python 目录，修改 snd.py 文件，编写对接设备时需要的信息，如连接信息、驱动信息等，具体信息参考《开放可编程开发指南》中“驱动包”章节，管理 NE40E-X3 设备的完整代码如下：

```
from aoc.snd.netconfsnd import NetconfSND
```

```
from aoc.snd.snd_model_pb2.sysoidinfo_pb2 import SysoidInfo
from aoc.snd.snd_model_pb2.connectinfo_pb2 import ConnectInfos, ProtocolEntity, DEFAULT_CONNECT,
PRIMARY_CONNECTION, \
    HelloEntity
from aoc.snd.snd_model_pb2.channelInfo_pb2 import SINGLE_CHANNEL
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import CommonDriverInfo, NetconfDriverInfo


class SND(NetconfSND):
    def getSysoidInfo(self, aoccontext, request=None):
        self.logger.info('getSysoidInfo start.')
        sysoidInfo = SysoidInfo()
        sysoidEntity = sysoidInfo.sysoidEntity.add()
        sysoidEntity.sysoid = "1.3.6.1.4.1.2011.2.62.2.8"
        sysoidEntity.deviceType = "ROUTER"
        sysoidEntity.deviceModel = "NE40E-X3"
        sysoidEntity.deviceVendor = "HUAWEI"
        self.logger.info('getSysoidInfo end.')
        return sysoidInfo

    def getConnectInfo(self, aoccontext, request=None):
        self.logger.info('getConnectInfo start.')
        connectInfos = ConnectInfos()
        connectInfo = connectInfos.connectInfo.add()
        connectInfo.protocolEntity.protocolType = ProtocolEntity.netconf
        connectInfo.protocolEntity.helloEntity.helloType = HelloEntity.extendType
        connectInfo.connectPolicy = DEFAULT_CONNECT
        connectInfo.channelInfo.readChannel = SINGLE_CHANNEL
        connectInfo.channelInfo.is_read_share_write = True
        connectInfo.connectionPriority = PRIMARY_CONNECTION
        self.logger.info('getConnectInfo end.')
        return connectInfos

    def getCommonDriverInfo(self, aoccontext, request=None):
        self.logger.info('getCommonDriverInfo start.')
        common_driver = CommonDriverInfo()
        common_driver.unsupportedOperations = "create,delete"
        common_driver.deleteStrategy = 1
        syncToDel = common_driver.para.add()
        syncToDel.key = "sync-to-del-enable"
        syncToDel.value = "true"
        self.logger.info('getCommonDriverInfo end.')
        return common_driver

    def getNetconfDriverInfo(self, aoccontext, request=None):
        self.logger.info('getNetconfDriverInfo start.')
        netconf_driver = NetconfDriverInfo()
        netconf_driver.phase = "two"
        netconf_driver.classification = "huawei-v5"
        #netconf_driver.testOption = "set"
        self.logger.info('getNetconfDriverInfo end.')
        return netconf_driver
```

修改后如图：



```
from aoc.snd.netconfsnd import NetconfSND

from aoc.snd.snd_model_pb2.sysoidinfo_pb2 import SysoidInfo
from aoc.snd.snd_model_pb2.connectinfo_pb2 import ConnectInfos, ProtocolEntity, DEFAULT_CONNECT, PRIMARY_CONNECTION
from aoc.snd.snd_model_pb2.channelInfo_pb2 import SINGLE_CHANNEL
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import CommonDriverInfo, NetconfDriverInfo


class SND(NetconfSND):
    def getSysoidInfo(self, aoccontext, request=None):
        self.logger.info('getSysoidInfo start.')
        sysoidInfo = SysoidInfo()
        sysoidEntity = sysoidInfo.sysoidEntity.add()
        sysoidEntity.sysoid = "1.3.6.1.4.1.2011.2.62.2.8"
        sysoidEntity.deviceType = "ROUTER"
        sysoidEntity.deviceModel = "NE40E-X3"
        sysoidEntity.deviceVendor = "HUAWEI"
        self.logger.info('getSysoidInfo end.')
        return sysoidInfo

    def getConnectInfo(self, aoccontext, request=None):
        self.logger.info('getConnectInfo start.')
        connectInfos = ConnectInfos()
        connectInfo = connectInfos.connectInfo.add()
        connectInfo.protocolEntity.protocolType = ProtocolEntity.netconf
        connectInfo.connectPolicy = DEFAULT_CONNECT
        connectInfo.channelInfo.readChannel = SINGLE_CHANNEL
        connectInfo.channelInfo.is_read_share_write = True
        connectInfo.connectionPriority = PRIMARY_CONNECTION
        self.logger.info('getConnectInfo end.')
        return connectInfos

    def getCommonDriverInfo(self, aoccontext, request=None):
```

代码解析：

1.引入头文件。

```
from aoc.snd.netconfsnd import NetconfSND

from aoc.snd.snd_model_pb2.sysoidinfo_pb2 import SysoidInfo
from aoc.snd.snd_model_pb2.connectinfo_pb2 import ConnectInfos, ProtocolEntity, DEFAULT_CONNECT, PRIMARY_CONNECTION, HelloEntity
from aoc.snd.snd_model_pb2.channelInfo_pb2 import SINGLE_CHANNEL
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import CommonDriverInfo, NetconfDriverInfo
```

导入本段代码中需要使用的模块。如果没有安装此模块，按照“实验环境说明”章节进行安装。

“NetconfSND”模块是所有网元驱动包的父类，默认的配置已在父类中实现，针对定制化的配置需要在网元驱动包中继承“NetconfSND”，覆写父类中的方法，下面步骤中会详细介绍；

“SysoidInfo”模块配置网元驱动包对应的设备类型、厂商、款型等信息，在设备纳管后根据该信息匹配对应网元驱动包；

“ConnectInfos”模块用于配置连接信息，如与设备的建连协议，建立几个通道，握手的 hello 报文等；

“CommonDriverInfo”模块用于定制设备配置报文中的一些操作类型，如设备不支持 create 需要用 merge；设备的同步方式是否是先删除后添加等；

“NetconfDriverInfo”模块配置 netconf 协议参数，如是否支持两阶段，是否支持 set 模式等。

2. 注册设备的 sysoid 信息（该部分代码用户只需修改设备 sysoid、设备类型、设备款型和设备厂商）。

```
def getSysoidInfo(self, aoccontext, request=None):
    self.logger.info('getSysoidInfo start.')
    sysoidInfo = SysoidInfo()
    sysoidEntity = sysoidInfo.sysoidEntity.add()
    sysoidEntity.sysoid = "1.3.6.1.4.1.2011.2.62.2.8"
    sysoidEntity.deviceType = "ROUTER"
```

```
    sysoidEntity.deviceModel = "NE40E-X3"
    sysoidEntity.deviceVendor = "HUAWEI"
    self.logger.info('getSysoidInfo end.')
    return sysoidInfo
```

定义 getSysoidInfo（self, aoccontext, request=None）方法，该方法定义网元驱动包中的 sysoid，如设备类型 deviceType、设备款型 deviceModel、设备厂商 deviceVendor 等。

设备上获取相应信息的办法：

- 获取 Sysoid

```
<NE40X3>system
[~NE40X3]diagnose
[~NE40X3-diagnose]display system information
  The system object ID:
1.3.6.1.4.1.2011.2.62.2.8
```

- DeviceType 默认为‘ROUTER’
- DeviceModel 为设备型号和 Vendor

```
<NE40X3>display version
Huawei Versatile Routing Platform Software
VRP (R) software, Version 8.150 (NE40E V800R009C10SPC100)
Copyright (C) 2012-2017 Huawei Technologies Co., Ltd.
HUAWEI NE40E-X3 uptime is 142 days, 7 hours, 48 minutes
```

3. 配置设备的连接能力信息（该部分可以直接调用，用户不需要修改）。

```
def getConnectInfo(self, aoccontext, request=None):
    self.logger.info('getConnectInfo start.')
    connectInfos = ConnectInfos()
    connectInfo = connectInfos.connectInfo.add()
    connectInfo.protocolEntity.protocolType = ProtocolEntity.netconf
    connectInfo.protocolEntity.helloEntity.helloType = HelloEntity.extendType
    connectInfo.connectPolicy = DEFAULT_CONNECT
    connectInfo.channelInfo.readChannel = SINGLE_CHANNEL
    connectInfo.channelInfo.is_read_share_write = True
    connectInfo.connectionPriority = PRIMARY_CONNECTION
    self.logger.info('getConnectInfo end.')
    return connectInfos
```

定义 getConnectInfo（self，aoccontext，request=None）方法，该方法覆写父类中的 getConnectInfo 方法，定义连接信息，定义协议类型 protocolType，握手 hello 报文类型 helloType，连接策略 connectPolicy，定义通道个数 channelInfo。

4. 配置设备类型（该部分代码可以直接调用，用户不需要修改）。

```
def getCommonDriverInfo(self, aoccontext, request=None):
    self.logger.info('getCommonDriverInfo start.')
    common_driver = CommonDriverInfo()
    common_driver.unsupportedOperations = "create,delete"
    common_driver.deleteStrategy = 1
    syncToDel = common_driver.para.add()
    syncToDel.key = "sync-to-del-enable"
    syncToDel.value = "true"
    self.logger.info('getCommonDriverInfo end.')
    return common_driver
```

```
def getNetconfDriverInfo(self, aoccontext, request=None):
    self.logger.info('getNetconfDriverInfo start.')
    netconf_driver = NetconfDriverInfo()
    netconf_driver.phase = "two"
    netconf_driver.classification = "huawei-v5"
    self.logger.info('getNetconfDriverInfo end.')
    return netconf_driver
```

定义 getCommonDriverInfo(self, aoccontext, request=None)方法，该方法用于定义与设备的交互策略：

- “common_driver.unsupportedOperations = "create,delete"”：设配不支持的操作类型，框架会将 create 换成 merge，delete 换成 remove；
- “common_driver.deleteStrategy = 1”：定义一种删除策略；
- “syncToDel.key = "sync-to-del-enable"”：定义 key 值；
- “syncToDel.value = "true"”：设置数据对账是否支持删除南向设备配置实例。取值：true（支持）；

定义 getNetconfDriverInfo(self, aoccontext, request=None)方法，该方法用于定制 netconf 协议参数

- “netconf_driver.phase = "two"”：设置 netconf 两阶段下发；
- “netconf_driver.classification = "huawei-v5"”：使用 yang 模型通道，本例为 huawei-v5。

1.4.5 配置秘钥

使用 Gpg4win 工具生成公钥和私钥。将私钥保存在驱动包中，将公钥上传到 iMaster NCE 上用于加密认证。

1.生成公钥和私钥。

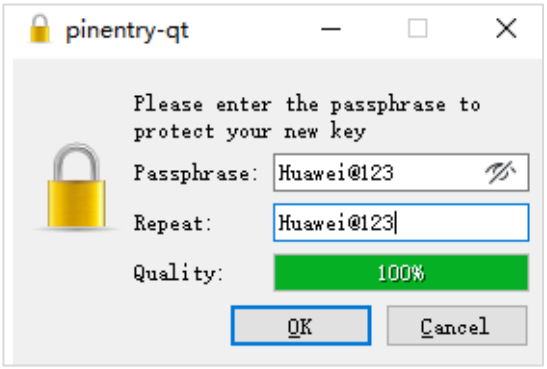
打开 Gpg4win 密钥生成工具，单击“新建密钥对”。

在“密钥创建向导”中输入姓名和电子邮件，单击“高级设置”，调整密钥参数。

单击“高级设置”，在对话框中选择“3072 比特”，单击“OK”，完成密钥参数的调整。

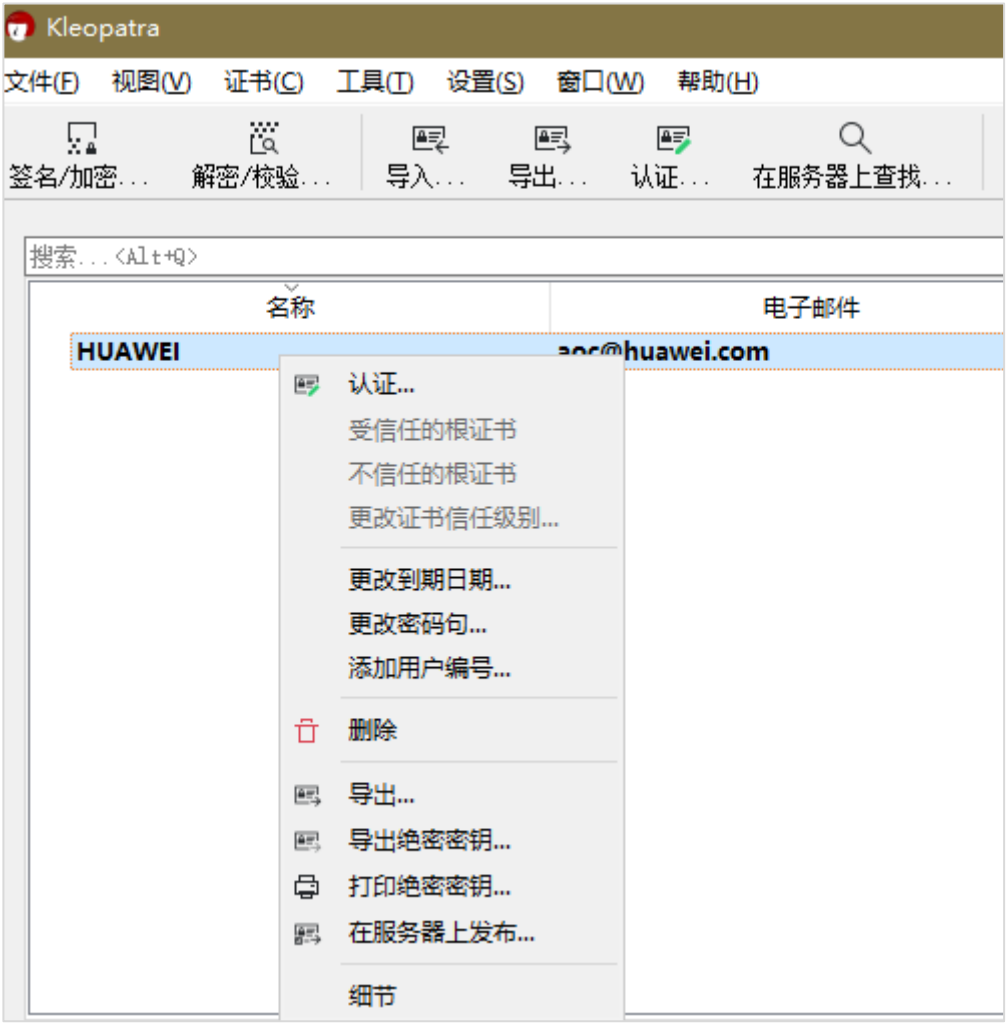
单击“下一步”，勾选“显示全部细节”，单击“新建”。

在弹框中设置密码，例如“Huawei@123”，单击“OK”，完成密钥创建。



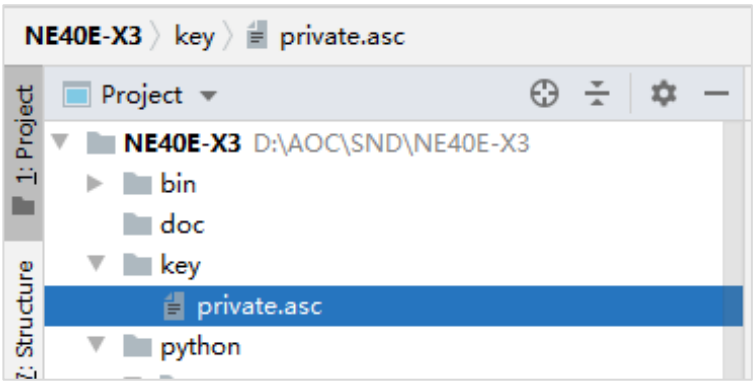
密钥创建成功后，单击“生成您的密钥对的副本”，导出密钥（命名 private.asc）。导出时需要输入密码验证。

在主页面中右击“导出”公钥文件（命名 public.asc），用于上传到 AOC 认证使用。



2.本地保存私钥。

将导出的密钥文件 private.asc 复制到 NE40E-X3 模板的 key 目录下。



3.上传公钥。

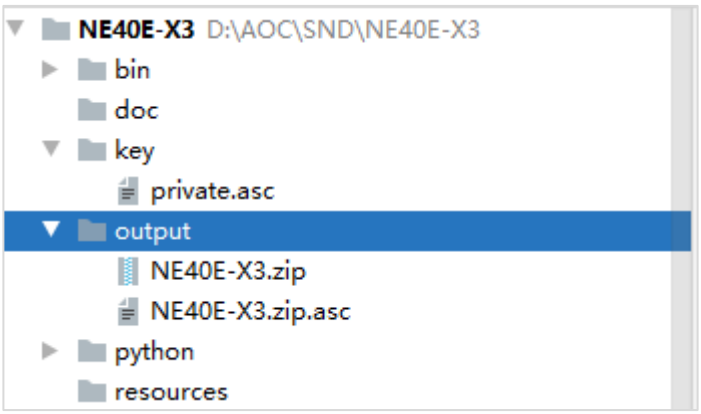
在 iMaster NCE“工程管理”>“公钥管理”中导入公钥。

1.4.6 生成网元驱动包

使用 CMD 切换到 NE40E-X3 模板的 bin 目录，运行 makeFile.bat 文件，输入已设置的密钥密码，本例为“Huawei@123”。

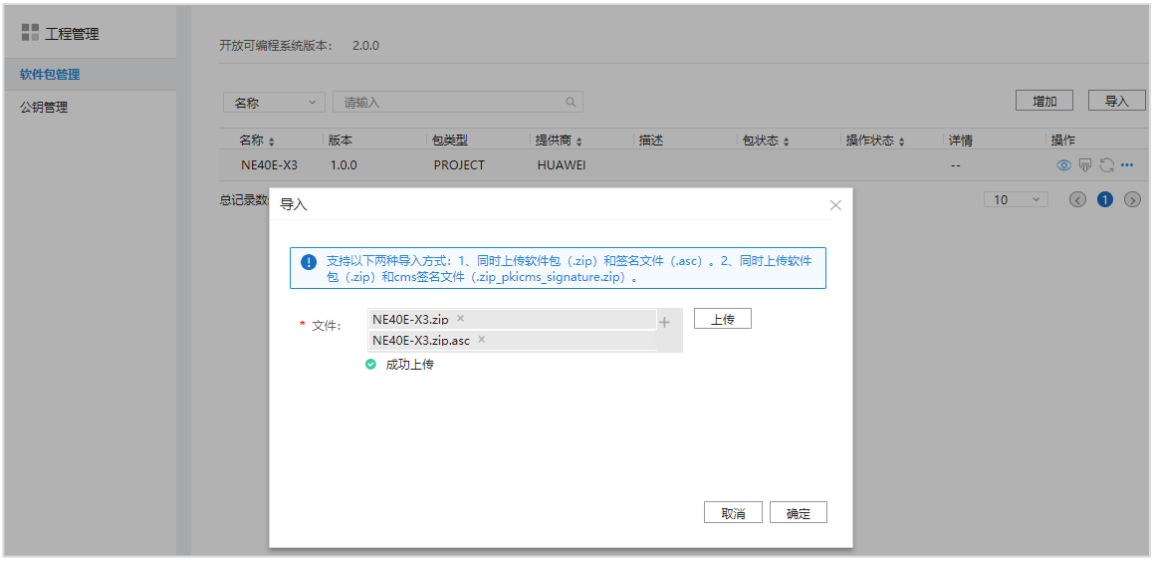
```
D:\AOC\SND\NE40E-X3>cd bin
D:\AOC\SND\NE40E-X3\bin>makeFile.bat
Please input password for private key:
…..
2020-07-06 14:40:58,975 INFO [com.huawei.ncecommon.extended.pkg.mgr.tools.common.FileUtil] - [Sign]Generate signature file success.
2020-07-06 14:40:58,977 INFO [com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign] Sign: Execute success
```

回显信息包含“sucess”即成功生成网元驱动包在 NE40E-X3 模板的 output 目录下。

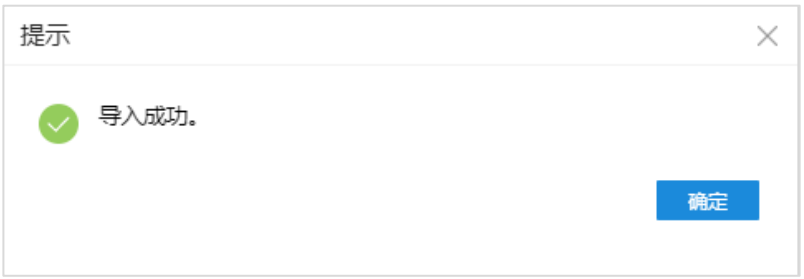


1.4.7 上传并激活网元驱动包

在 iMaster NCE“工程管理”>“软件包管理”中导入网元驱动包。



点击“确定”，导入成功。



单击右侧“操作”中的“安装”按钮，激活驱动包。

1.5 使用 iMaster NCE 纳管设备

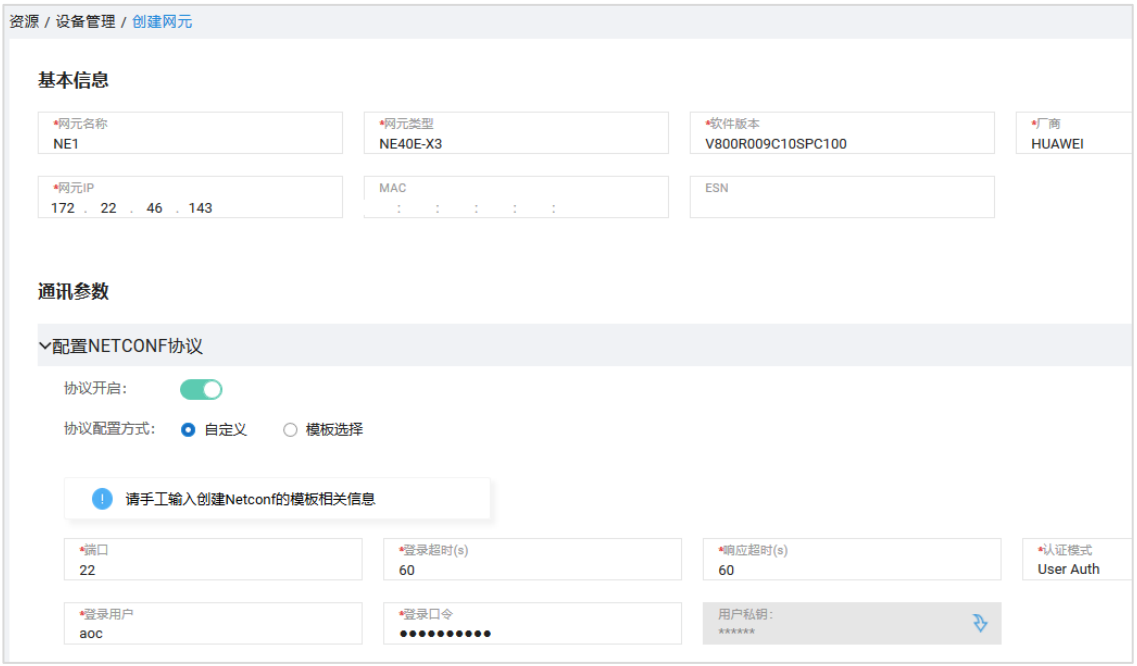
1.5.1 获取设备对接信息

在“沙箱预约”拓扑中获取 NE40E 设备的相关信息。本例中 IP 地址为 172.22.46.143，NETCONF 用户名为“aoc”，密码为“Huawei@123”。

1.5.2 纳管设备

进入 iMaster NCE“资源”下的“设备管理”页面。

单击“创建”，进入“创建网元”页面，填写基本信息，其中网元类型、软件版本、厂商要与网元驱动包中 devices 信息相同，设备就是根据网元类型、软件版本和厂商这些关键信息找到对应的网元驱动包。

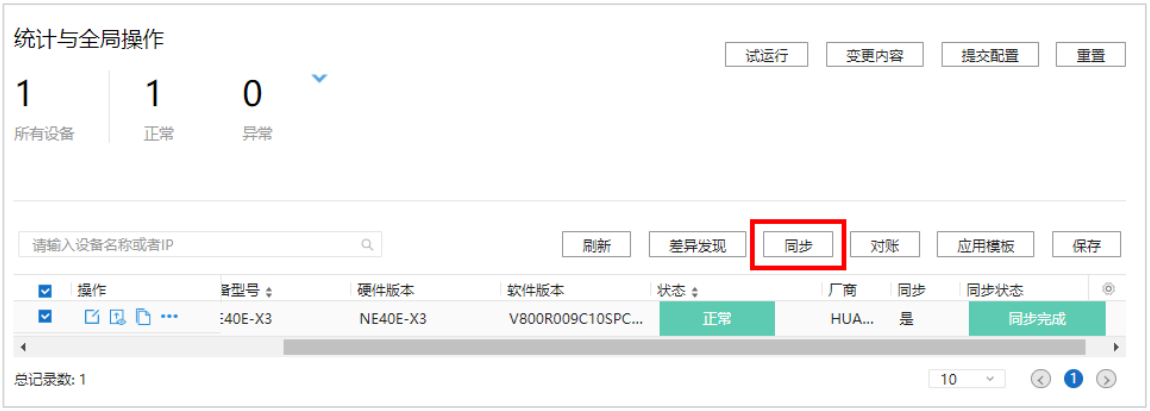


本例中信息参考 1.4.2 包配置文件参数。

最后选择“配置管理>网元配置>设备管理”，可以查看新添加的设备。

1.5.3 同步设备信息

选择被纳管设备，点击“同步”。最后“同步状态”显示“同步完成”。



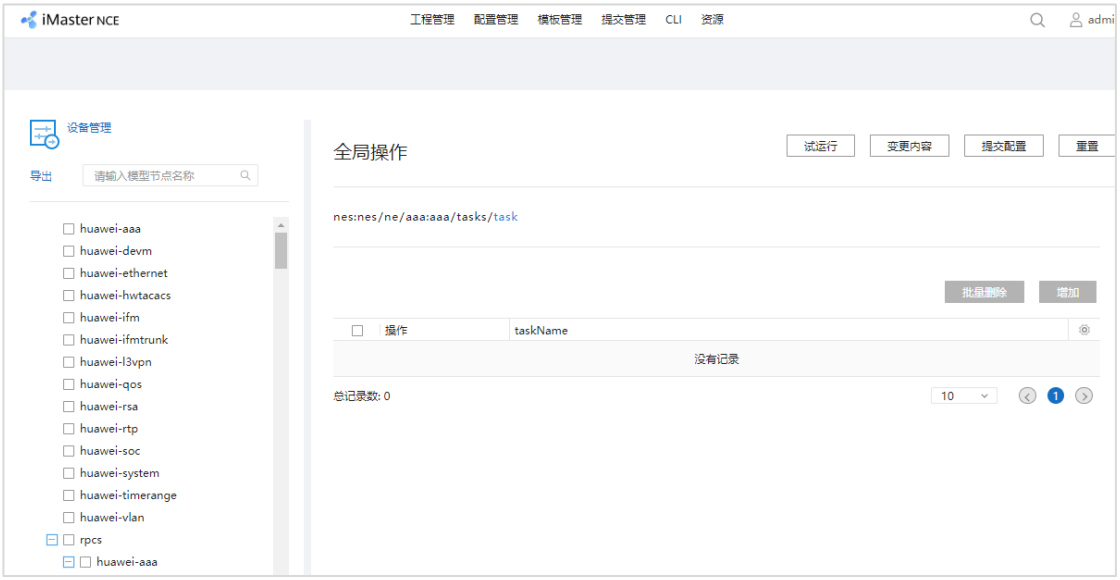
1.6 使用界面下发设备配置

成功纳管好设备后，可以基于 iMaster NCE 页面实现设备原生能力的配置下发。本小结将介绍在 NE40E-X3 设备上创建子接口。

1.进入设备单站页面。

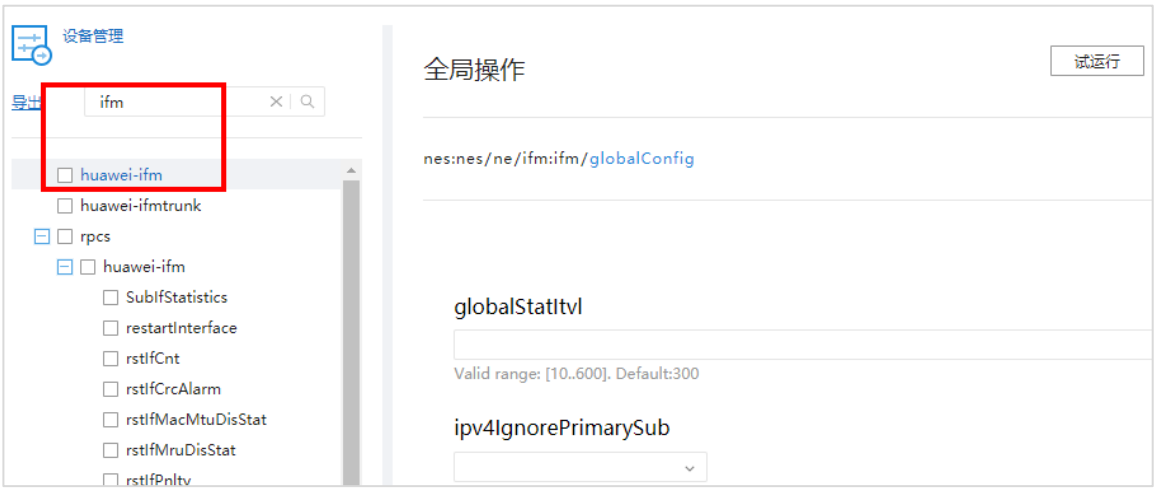
单击设备列表中，设备后面“操作”中“编辑”按钮。

进入单站页面。

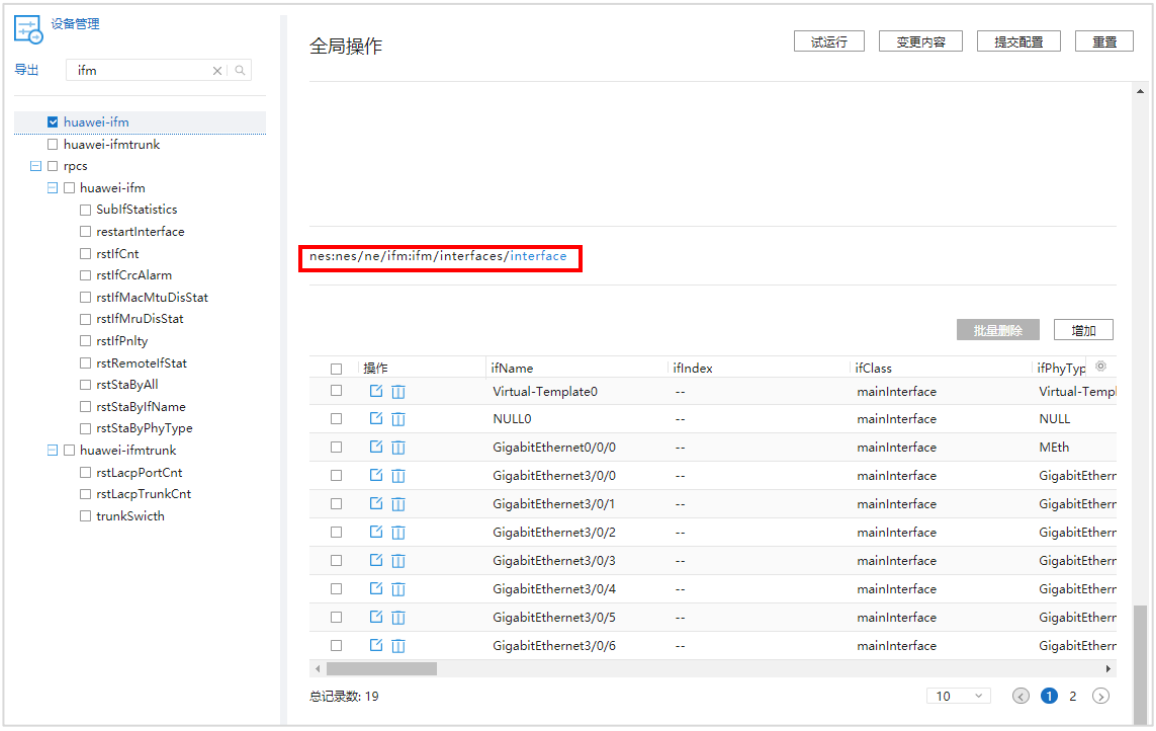


2.配置创建子接口业务。

创建子接口对应的 YANG 文件为“huawei-ifm”。左侧搜索 ifm。



右侧滚动栏下滑到“nes:nes/ne/ifm:ifm/interfaces/interface”。

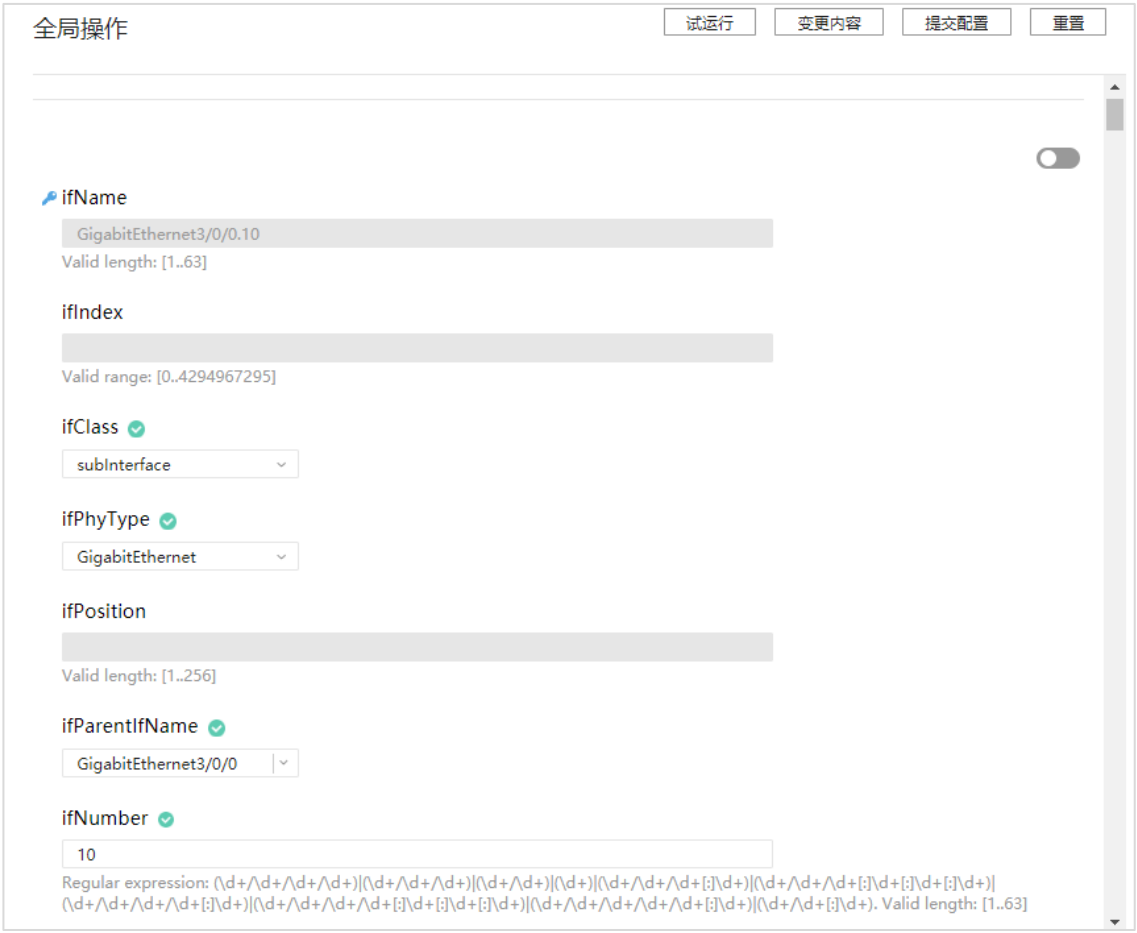


在完成 1.5.3“同步设备信息”步骤后会如图显示出设备当前所有接口信息。

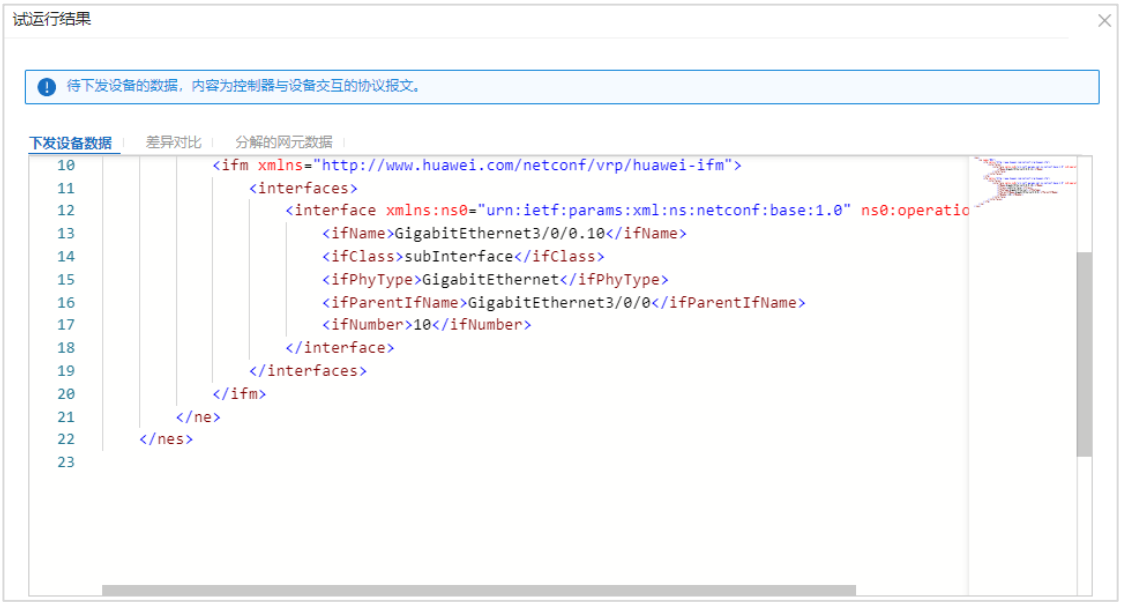
本节示例创建子接口，例如创建 GigabitEthernet3/0/0.10。点击“增加”。

ifName 中输入完整的“GigabitEthernet3/0/0.10”，点击“创建”。

然后在“ifclass”、“ifPhyType”和“ifParentIfName”下拉菜单中选择“subinterface”、“GigabitEthernet”和“GigabitEthernet3/0/0”，在“ifNumber”下填写“10”。



3.单击“试运行”，提前查看将要下发的配置报文，检查是否正确。



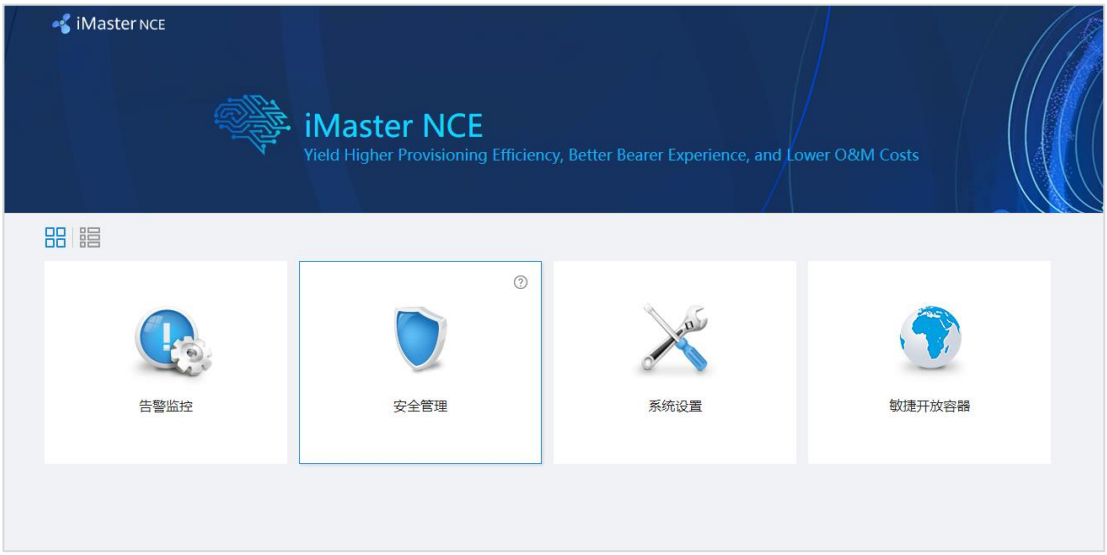
4.单击“提交配置”，将配置下发到设备上。

1.7 通过北向接口下发设备配置

iMaster NCE 可以基于设备原生能力，提供北向开放开放能力。首先创建北向用户，然后使用北向账户实现“增删改查”。更多 iMaster NCE 北向能力请参考开发者社区“资源下载”《数通网络开放可编程北向 Rest+API 参考》。

1.7.1 创建北向用户

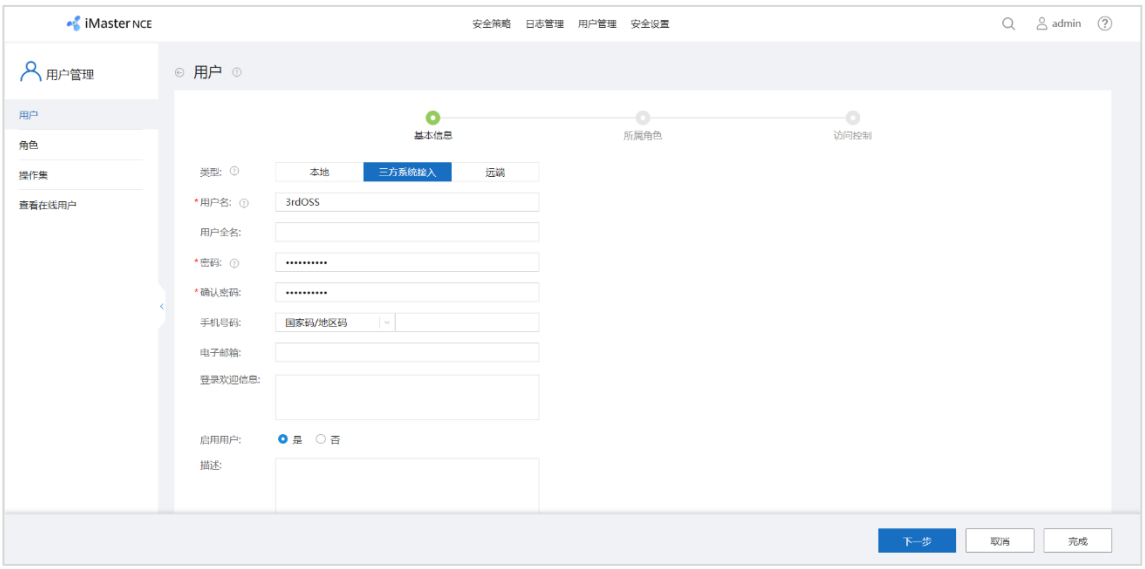
iMaster NCE 主界面单击“安全管理”。



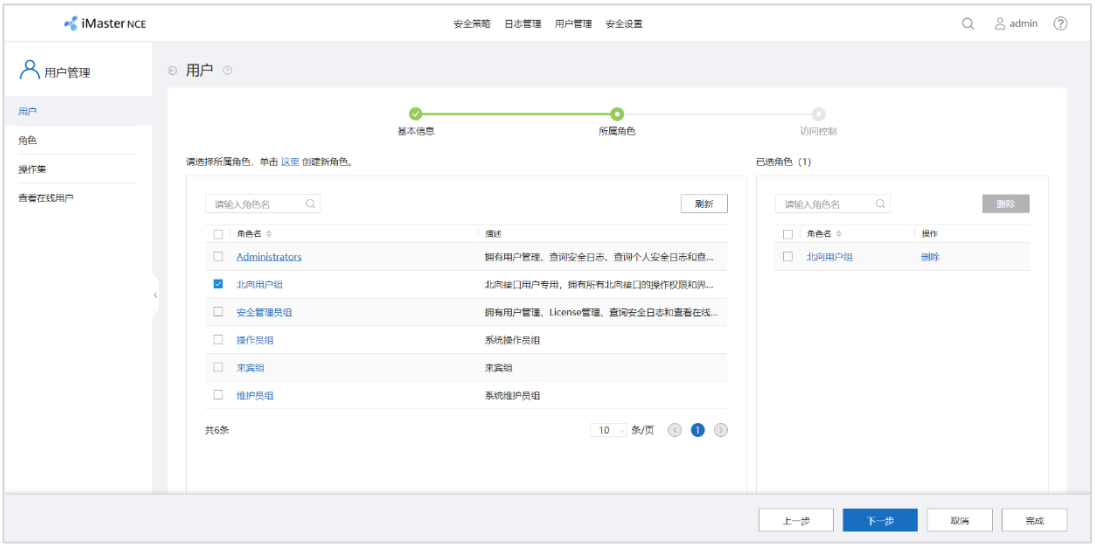
进入安全管理页面，单击“用户管理”按钮，单击“创建”。

进入创建用户页面，选择“三方系统接入”，添加用户名及密码，单击下一步 。

本例中创建用户名为“3rdOSS”，密码为“Huawei@123”。



勾选‘北向用户组’，单击“下一步”，后续按向导默认配置完成用户创建。



1.7.2 创建配置

本章节介绍实验相关基础的北向操作，更多信息参考《数通网络开放可编程用户指南》“北向API”章节。

1.获取 token

Method：PUT

```
https://{{northIP}}:26335/rest/plat/smapp/v1/oauth/token
```

请求头 Header

Key	Value
Content-Type	application/json
Accept	application/json

请求体

```
{
    "grantType":"password",
    "userName":"3rdOSS",
    "value":"Huawei@123"
}
```

返回体样例

```
{      "accessSession": "x-
1cpceq1j0bdhft2niqiotg087uak6ps82ptfuqnw2pteqm84ob04o77u3vo4rtlg6mpdle2ktgo9dfml9cmrbzljs5u
k2k1f9ho7rw2pen6kqk49obdcamc8jzvs6llj",
      "roaRand": "48fa2a40d3250e6beb6a16cd806034574cf4fd9969a5f096",
      "expires": 1800,
      "additionalInfo": null
}
```

返回报文中的 accessSession 就是 token，下面所有发送的报文头中都需要带上 token，进行鉴权。

2.申请事务（对业务进行编辑前，需要申请事务 ID。事务管理支持配置变化支持在一个原子事务里提交，保证 NCE 的数据和转发器的数据保持一致性）

Method：POST

```
https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:create
```

请求头 Header，请求头中包含步骤 3 中生成的 token

Key	Value
Content-Type	application/json
Accept	application/json
Cookie	accessSession={{accessSession}}

返回体样例，其中 trans-id 就是新建的事务 id：

```
{
     "huawei-ac-restconf-transactions:output": {
          "trans-id": "a4ce2a02-5e6f-41af-8824-852ba2155c2b"
     }
}
```

3.编辑下发的配置（以下发 VLAN 为例）

Method：POST

```
https://{{northIP}}:26335/restconf/v1/data/huawei-ac-nes:inventory-cfg/nes/ne/d4acf7f0-6e3c-11ea-
a6ea-3e544f98513d/huawei-vlan:vlan/vlans
```

请求头 Header 中携带步骤 1 生成的 token 和步骤 2 生成的事务 id。

Key	Value
restconf-transaction-id	{{transactionsID}}
Content-Type	application/json

Accept	application/json
Cookie	accessSession={{accessSession}}

请求体 Body

```
{
    "vlan": [
        {
            "vlanId" : "202",
            "vlanName" : "testAOC202",
            "vlanDesc" : "testAOC"
        }
    ]
}
```

4.试运行

Method：POST

```
https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:dry-run
```

请求头 Header

Key	Value
Content-Type	application/json
Accept	application/json
Cookie	accessSession={{accessSession}}

请求体

```
{
    "huawei-ac-restconf-transactions:input":{
        "trans-id":{{transactionsID}}
    }
}
```

返回体样例：

```
{
    "huawei-ac-restconf-transactions:output": {
        "result": true
    }
}
```

备注：编辑完业务后，试运行操作，用于提前查看将要下发的设备，但是不下发到设备。

5.配置差异预览

Method：POST

```
https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:diff
```

请求头 Header

Key	Value

Content-Type	application/json
Accept	application/json
Cookie	accessSession={{accessSession}}

请求体

```
{
    "huawei-ac-restconf-transactions:input":{
        "trans-id": {{transactionsID}}
    }
}
```

返回体

```
{
    "huawei-ac-restconf-transactions:output": {
        "ne-diff-info": [
            {
                "ne-id": "f3874dce-7a04-11ea-802c-f64ba7e1c586",
                "ne-name": "NE1",
                "diff-infos": [
                    {
                        "feature": "/(http://www.huawei.com/netconf/vrp/huawei-vlan?revision=2018-
06-11)vlan",
                        "diff-info": "{\"vlan\": {\"vlans\": {\"vlan\": [{\"left\": null, \"right\":
{\"[vlanId=202]\": {\"vlanId\": 202, \"vlanName\": \"testAOC202\", \"vlanDesc\": \"testAOC\"}}}]}}}"
                    }
                ]
            }
        ],
        "service-diff-infos": [
            {}
        ]
    }
}
```

编辑完配置后，预览当前事务的修改内容，返回网元层数据的差异。

6.查看试运行后的网元配置

Method：POST

```
https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:dry-run-query
```

请求头 Header

Key	Value
Content-Type	application/json
Accept	application/json
Cookie	accessSession={{accessSession}}

请求体

```
{
```

```
    "huawei-ac-restconf-transactions:input":{
        "trans-id": {{transactionsID}}
    }
}
```

返回体

```
{
    "huawei-ac-restconf-transactions:output": {
        "native": {
            "dry-run-ne-native-confs-item": [
                {
                    "ne-id": "f3874dce-7a04-11ea-802c-f64ba7e1c586",
                    "ne-name": "NE1",
                    "ne-cfg": [
                        "<vlan xmlns=\"http://www.huawei.com/netconf/vrp/huawei-
vlan\">\n<vlans>\n<vlan xmlns:ns0=\"urn:ietf:params:xml:ns:netconf:base:1.0\"
ns0:operation=\"merge\">\n<vlanId>202</vlanId>\n<vlanName>testAOC202</vlanName>\n<vlanDesc>t
estAOC</vlanDesc>\n</vlan>\n</vlans>\n</vlan>\n"
                    ]
                }
            ]
        },
        "diff": {
            "dry-run-ne-diff-confs-item": [
                {
                    "ne-id": "f3874dce-7a04-11ea-802c-f64ba7e1c586",
                    "ne-name": "NE1",
                    "ne-cfg-diffs": [
                        {
                            "feature": "/(http://www.huawei.com/netconf/vrp/huawei-
vlan?revision=2018-06-11)vlan",
                            "diff-info": "{\"vlan\": {\"vlans\": {\"vlan\": [{\"left\": null, \"right\":
{\"[vlanId=202]\": {\"vlanId\": 202, \"vlanName\": \"testAOC202\", \"vlanDesc\": \"testAOC\"}}}]}}}"
                        }
                    ]
                }
            ]
        },
        "mapconf": {}
    }
}
```

查看试运行后生成的配置和差异信息。

7.提交配置

Method：POST

```
https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:commit
```

请求头 Header

Key	Value
Content-Type	application/json
Accept	application/json

Cookie	accessSession={{accessSession}}

请求体

```
{
    "huawei-ac-restconf-transactions:input":{
        "force":false,
        "trans-id": {{transactionsID}},
        "no-network":true
    }
}
```

返回体

```
{
    "huawei-ac-restconf-transactions:output": {
        "result": true,
        "reason": ""
    }
}
```

业务编辑结束后，将编辑的数据提交到控制器上，使其正式生效。

1.7.3 查看配置

1.获取 token

Method：PUT

```
https://{{northIP}}:26335/rest/plat/smapp/v1/oauth/token
```

请求头 Header

Key	Value
Content-Type	application/json
Accept	application/json

请求体

```
{
   "grantType":"password",
   "userName":"3rdOSS",
   "value":"Huawei@123"
}
```

返回体样例

```
{     "accessSession": "x-
1cpceq1j0bdhft2niqiotg087uak6ps82ptfuqnw2pteqm84ob04o77u3vo4rtlg6mpdle2ktgo9dfml9cmrbzljs5u
k2k1f9ho7rw2pen6kqk49obdcamc8jzvs6llj",
    "roaRand": "48fa2a40d3250e6beb6a16cd806034574cf4fd9969a5f096",
    "expires": 1800,
    "additionalInfo": null
}
```

返回报文中的 `accessSession` 就是 token，下面所有发送的报文头中都需要带上 token，进行鉴权。

2.查看配置

Method：GET

```
https://{{northIP}}:26335/restconf/v1/data/huawei-ac-nes:inventory-cfg/nes/ne/e76d5b7b-7936-11ea-802c-f64ba7e1c586/huawei-vlan:vlan/vlans/vlan/202?offset=2&limit=10
```

请求头 Header

Key	Value
Accept	application/json
Cookie	accessSession={{accessSession}}
Content-Type	application/json

返回体

```
{
    "vlan": [
        {
            "vlanId": 202,
            "vlanName": "testAOC202",
            "vlanDesc": "testAOC"
        }
    ]
}
```

1.7.4 修改配置

1.获取 token

Method：PUT

```
https://{{northIP}}:26335/rest/plat/smapp/v1/oauth/token
```

请求头 Header

Key	Value
Content-Type	application/json
Accept	application/json

请求体

```
{
    "grantType":"password",
    "userName":"3rdOSS",
    "value":"Huawei@123"
}
```

返回体样例

```
{      "accessSession": "x-
1cpceq1j0bdhft2niqiotg087uak6ps82ptfuqnw2pteqm84ob04o77u3vo4rtlg6mpdle2ktgo9dfml9cmrbzljs5u
k2k1f9ho7rw2pen6kqk49obdcamc8jzvs6llj",
      "roaRand": "48fa2a40d3250e6beb6a16cd806034574cf4fd9969a5f096",
      "expires": 1800,
      "additionalInfo": null
}
```


返回报文中的 accessSession 就是 token，下面所有发送的报文头中都需要带上 token，进行鉴权。

2.申请事务

Method：POST

```
https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:create
```

请求头 Header，请求头中包含步骤三中生成的 token

Key	Value
Content-Type	application/json
Accept	application/json
Cookie	accessSession={{accessSession}}

返回体样例，其中 trans-id 就是新建的事务 id：


```
{
      "huawei-ac-restconf-transactions:output": {
            "trans-id": "a4ce2a02-5e6f-41af-8824-852ba2155c2b"
      }
}
```


3.修改配置

Method：PUT

```
https://{{northIP}}:26335/restconf/v1/data/huawei-ac-nes:inventory-cfg/nes/ne/e76d5b7b-7936-11ea-
802c-f64ba7e1c586/huawei-vlan:vlan/vlans/vlan/202?offset=2&limit=10
```

请求头 Header

Key	Value
restconf-transaction-id	{{transactionsID}}
Accept	application/json
Cookie	accessSession={{accessSession}}
Content-Type	application/json

请求体

```
{
      "vlan": [
          {
                "vlanId" : "202",
```

```
                "vlanName" : "testAOC202",
                "vlanDesc" : "testAOC222"
            }
        ]
}
```

4.试运行

Method：POST

```
https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:dry-run
```

请求头 Header

Key	Value
restconf-transaction-id	{{transactionsID}}
Accept	application/json
Cookie	accessSession={{accessSession}}
Content-Type	application/json

请求体

```
{
    "huawei-ac-restconf-transactions:input":{
        "trans-id":{{transactionsID}}
    }
}
```

返回体样例

```
{
    "huawei-ac-restconf-transactions:output": {
        "result": true
    }
}
```

5.配置差异预览

Method：POST

```
https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:diff
```

请求头 Header

Key	Value
Content-Type	application/json
Accept	application/json
Cookie	accessSession={{accessSession}}

请求体

```
{
    "huawei-ac-restconf-transactions:input":{
        "trans-id": {{transactionsID}}
```

```
    }
}
```

返回体

```
{
    "huawei-ac-restconf-transactions:output": {
        "ne-diff-info": [
            {
                "ne-id": "f3874dce-7a04-11ea-802c-f64ba7e1c586",
                "ne-name": "NE1",
                "diff-infos": [
                    {
                        "feature": "/(http://www.huawei.com/netconf/vrp/huawei-vlan?revision=2018-06-11)vlan",
                        "diff-info": "{\"vlan\": {\"vlans\": {\"vlan\": [{\"[vlanId=202]\": {\"vlanId\": 202, \"vlanDesc\": {\"left\": \"testAOC\", \"right\": \"testAOC222\"}}}]}}}"
                    }
                ]
            }
        ],
        "service-diff-infos": [
            {}
        ]
    }
}
```

编辑完配置后，预览当前事务的修改内容，返回网元层数据的差异。

6.查看试运行后的网元配置

Method：POST

```
https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:dry-run-query
```

请求头 Header

Key	Value
Content-Type	application/json
Accept	application/json
Cookie	accessSession={{accessSession}}

请求体

```
{
    "huawei-ac-restconf-transactions:input":{
        "trans-id": {{transactionsID}}
    }
}
```

返回体

```
{
    "huawei-ac-restconf-transactions:output": {
        "native": {
            "dry-run-ne-native-confs-item": [
```

```
                    {
                        "ne-id": "f3874dce-7a04-11ea-802c-f64ba7e1c586",
                        "ne-name": "NE1",
                        "ne-cfg": [
                            "<vlan xmlns=\"http://www.huawei.com/netconf/vrp/huawei-
vlan\">\n<vlans>\n<vlan xmlns:ns0=\"urn:ietf:params:xml:ns:netconf:base:1.0\"
ns0:operation=\"merge\">\n<vlanId>202</vlanId>\n<vlanDesc>testAOC222</vlanDesc>\n</vlan>\n</vla
ns>\n</vlan>\n"
                        ]
                    }
                ]
            },
            "diff": {
                "dry-run-ne-diff-confs-item": [
                    {
                        "ne-id": "f3874dce-7a04-11ea-802c-f64ba7e1c586",
                        "ne-name": "NE1",
                        "ne-cfg-diffs": [
                            {
                                "feature": "/(http://www.huawei.com/netconf/vrp/huawei-
vlan?revision=2018-06-11)vlan",
                                "diff-info": "{\"vlan\": {\"vlans\": {\"vlan\": [{\"[vlanId=202]\": {\"vlanId\":
202, \"vlanDesc\": {\"left\": \"testAOC\", \"right\": \"testAOC222\"}}}]}}}"
                            }
                        ]
                    }
                ]
            },
            "mapconf": {}
        }
}
```


查看试运行后生成的配置和差异信息。

6.提交配置

Method：POST

```
https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:commit
```

请求头 Header

Key	Value
Content-Type	application/json
Accept	application/json
Cookie	accessSession={{accessSession}}

请求体

```
{
    "huawei-ac-restconf-transactions:input":{
        "force":false,
        "trans-id": {{transactionsID}},
        "no-network":true
    }
```

```
}
```

返回体

```
{
    "huawei-ac-restconf-transactions:output": {
        "result": true,
        "reason": ""
    }
}
```

1.7.5 删除配置

1.获取 token

Method：PUT

```
https://{{northIP}}:26335/rest/plat/smapp/v1/oauth/token
```

请求头 Header

Key	Value
Content-Type	application/json

请求体

```
{
   "grantType":"password",
   "userName":"3rdOSS",
   "value":"Huawei@123"
}
```

返回体

```
{
    "accessSession": "x-1cpceq1j0bdhft2niqiotg087uak6ps82ptfuqnw2pteqm84ob04o77u3vo4rtlg6mpdle2ktgo9dfml9cmrbzljs5uk2k1f9ho7rw2pen6kqk49obdcamc8jzvs6llj",
    "roaRand": "48fa2a40d3250e6beb6a16cd806034574cf4fd9969a5f096",
    "expires": 1800,
    "additionalInfo": null
}
```

获取 token，下面所有发送的报文头中都需要带上 token。

2.申请事务

Method：POST

```
https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:create
```

请求头 Header，申请事务 id，请求头中包含步骤三中生成的 token。

Key	Value
Content-Type	application/json

Accept	application/json
Cookie	accessSession={{accessSession}}

返回体中返回事务 id

```
{
    "huawei-ac-restconf-transactions:output": {
        "trans-id": "a4ce2a02-5e6f-41af-8824-852ba2155c2b"
    }
}
```

对业务进行编辑前，需要申请事务 ID。

3.删除配置

Method：DELETE

```
https://{{northIP}}:26335/restconf/v1/data/huawei-ac-nes:inventory-cfg/nes/ne/e76d5b7b-7936-11ea-802c-f64ba7e1c586/huawei-vlan:vlan/vlans/vlan/202?offset=2&limit=10
```

请求头 Header

Key	Value
restconf-transaction-id	{{transactionsID}}
Accept	application/json
Cookie	accessSession={{accessSession}}
Content-Type	application/json

返回体：200

1.8 思考题

1.如果被纳管设备升级，iMaster NCE 上的驱动包能否继续使用?

2.iMaster NCE 上需要同时添加多台设备，如何操作?

3.当设备不支持 NETCONF 对接时，如老的设备款型，只支持 CLI/SSH 对接，如何操作?

4.客户有自有的 OSS 系统如何对接控制器?

2 构建新业务-业务包实践

2.1 实验背景

华为 iMaster NCE 业务开放可编程基于 YANG 模型驱动的开放架构，以网元驱动包和业务包的形式使能网元层和网络层开放可编程，自动生成配置页面和北向接口，实现新设备的快速对接、新网络业务的快速构建和网络业务能力开放。

本实验将指导读者编写业务包（Specific Service Plugin，SSP）开放网络业务定义能力，即在 iMaster NCE 纳管网络设备后，读者自定义网络业务并生成北向接口。本课程您会学习：

- 根据业务需求编写业务包的流程
- 业务 YANG 文件编写
- Jinja2 模板编写

2.1.1 业务包介绍

业务包（SSP）是 iMaster NCE 软件包的一种，它定义了完成一套网络级业务配置对应的数据模型。该数据模型通常包含一个 Jinja2 模板文件、一个 Python 映射脚本和业务 YANG 模型。其中：

- Jinja2 模板描述了业务的数据结构，并使用 Jinja2 语法完成了诸如插值、条件判断、循环等操作。
- Python 映射脚本描述了如何将用户提交的数据填充到模板，并映射到网元数据结构中。
- 业务 YANG 模型描述了业务的相关参数，按照业务输入，构建业务 YANG 模型。

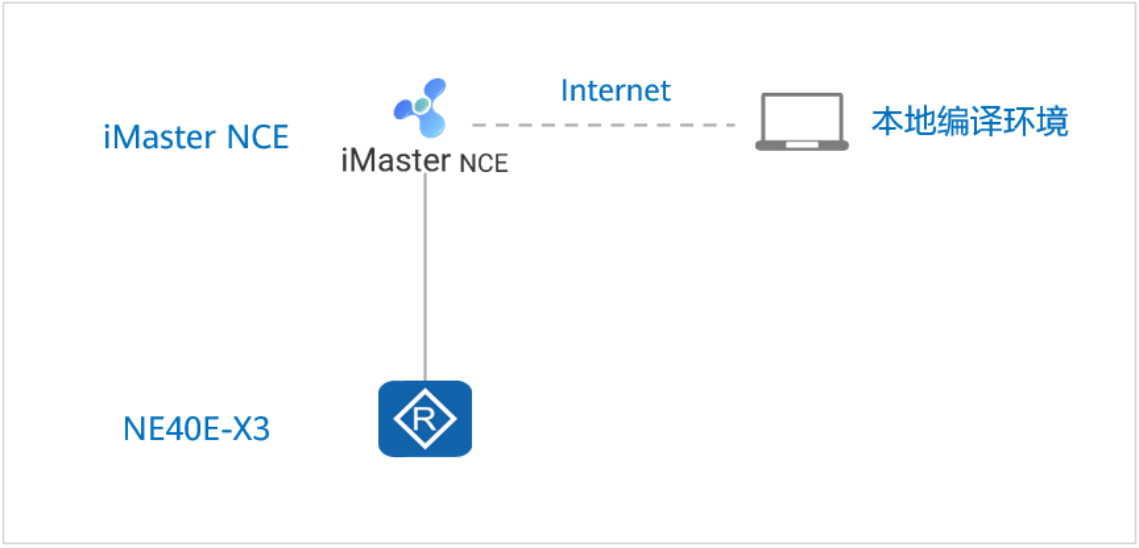
2.1.2 业务能力开放介绍

iMaster NCE 具备网络业务开放能力。基于业务 YANG 模型驱动，NCE 业务开放可编程系统自动生成南北向接口，快速构建新业务。使用 Easymap 算法，用户只需写创建流程，更新和删除都由算法比较计算得出，简化客户编程步骤，降低开发难度。

本实验将主要介绍 iMaster NCE 业务能力开放流程和操作，更多能力介绍和相关原理请学习 HCIP-Datacom-Network Automation Developer 教材《NCE 业务开放可编程》。

2.2 实验介绍

2.2.1 组网说明



本实验组网包含三个对象，NE40E-X3、iMaster NCE 和本地编译环境。其中 NE40E-X3、iMaster NCE 可由开发者社区提供沙箱环境，读者需在本地完成相关包的编译。

2.2.2 实验目标

本实验将自定义一个 VPN 网络业务，并北向开放此网络业务。

VPN 网络业务的对应的设配置逻辑为：创建一个 VPN 实例，创建一个子接口，将子接口绑定此 VPN 实例。根据此网络业务梳理出网络配置的输入参数如下：

参数名	参数值
ifName	GigabitEthernet3/0/0.20
ifClass	sub-interface
ifParentIfName	GigabitEthernet3/0/0
ifNumber	20
IfDescr	For_Test_AOC
ifMtu	200
vrfName	Test_AOC

最终实验的呈现效果为用户只需要输入对应参数的参数值，VPN 网络业务将自动在网络设备上开通。

注意：本实验在 iMaster NCE 已加载网元驱动包并成功纳管设备前提下进行。

2.2.3 实验步骤

本实验步骤如下：

1. 环境准备：准备本地环境和实验沙箱环境。
2. 本地编写业务包并加载到 iMaster NCE。
3. 通过 iMaster NCE 进行网络业务下发。
4. 通过 iMaster NCE 北向接口进行网络业务下发。

2.3 环境准备

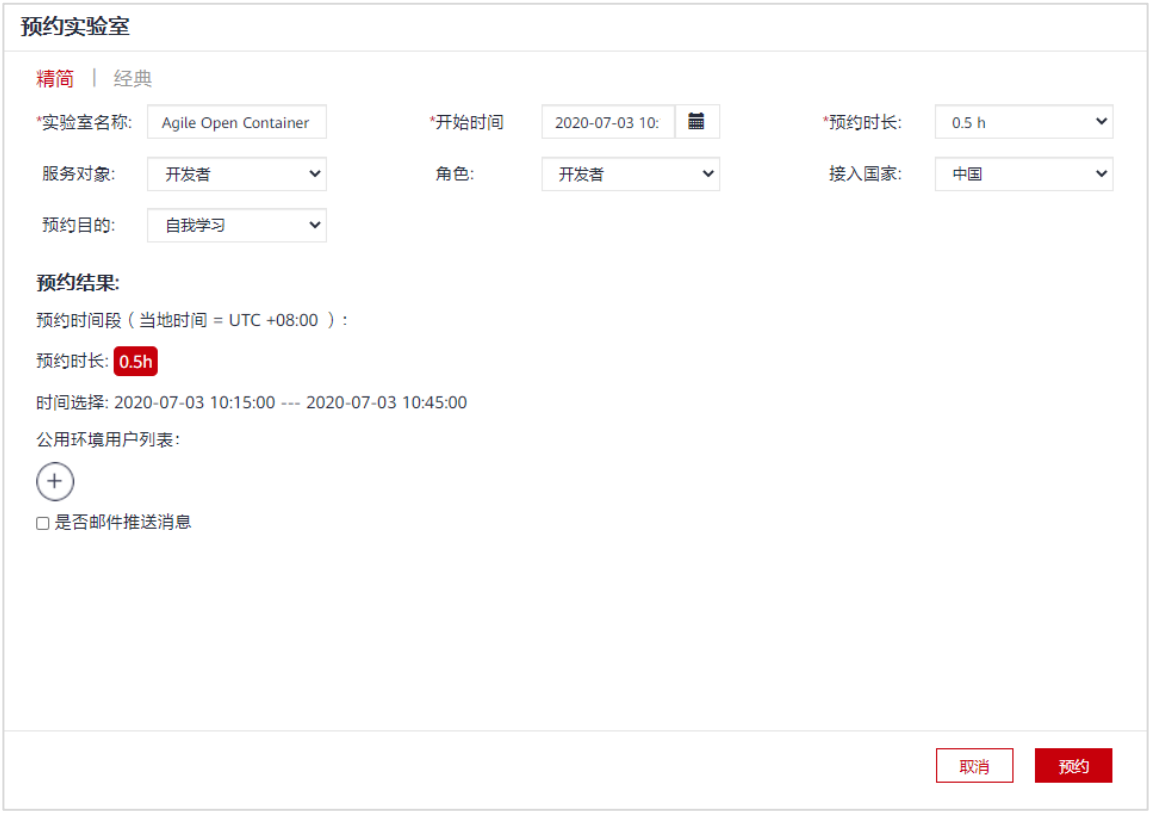
2.3.1 本地环境准备

本地环境准备参考“前言”>“实验环境说明”>“本地编译环境准备”。

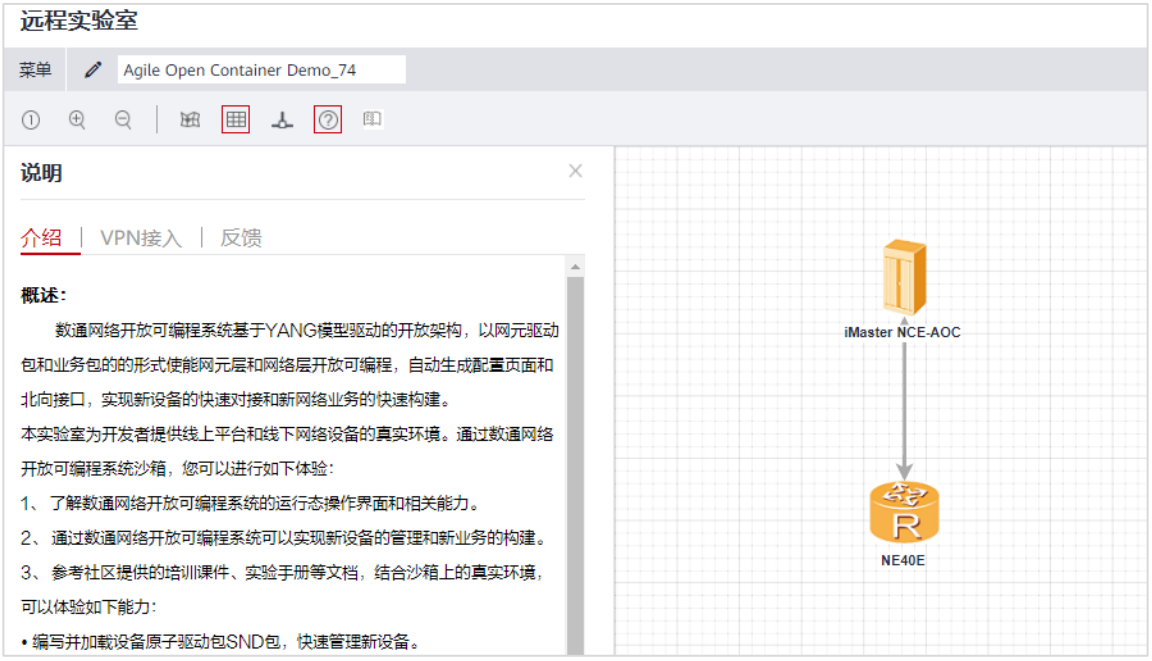
2.3.2 沙箱预约

开发者社区提供华为数通解决的沙箱体验，本实验申请在沙箱“远程实验室”中申请“Agile Open Container”，https://devzone.huawei.com/openecosystem/。

点击预约。



填写预约信息后即可进入沙箱环境。沙箱环境中包含拓扑和设备描述等详细信息。



2.3.3 iMaster NCE 纳管设备

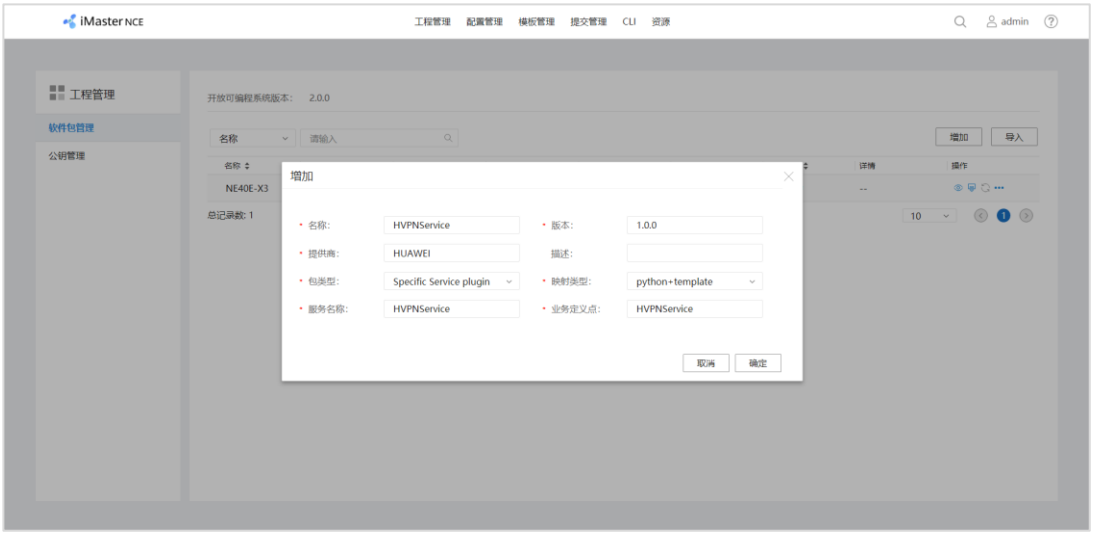
根据“网元驱动包实践”章节，读者编写网元驱动包并在 iMaster NCE 上纳管 NE40E-X3。

2.4 编写业务包（SSP）

本章节您将尝试在本地环境编写 NE40E 的业务包，并将其加载到 iMaster NCE。本实验的相关代码样例和资源文件可参考 https://devzone.huawei.com/apistudio/sample/aoc/apiSdk.html。

2.4.1 创建业务包模板

登录 iMaster NCE，进入“敏捷开放容器”。在“工程管理”页面，单击“添加”，填写业务包参数信息，填写完成之后单击“确定”按钮，完成创建模板。



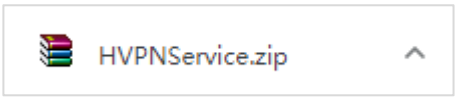
参数内容如下：

- 名称：HVPNService
- 版本：1.0.0
- 提供商：HUAWEI
- 包类型：Specific Service plugin
- 映射类型：python+template
- 服务名称：HVPNService
- 业务定义点：HVPNService

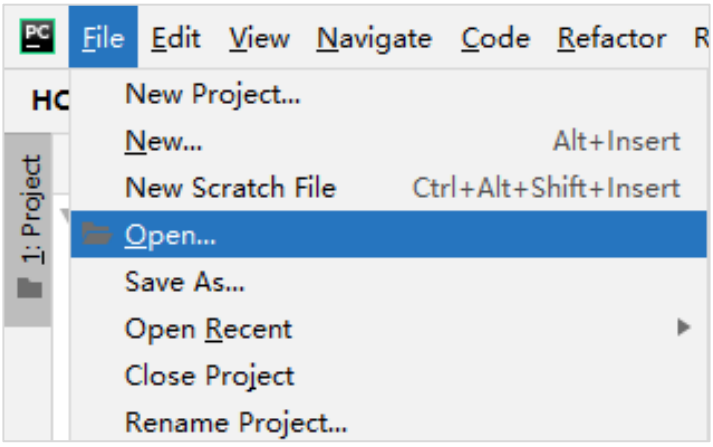
2.4.2 导出业务模板到本地 IDE

成功创建模板后，下载模板到本地。单击“操作”中的“导出”按钮。

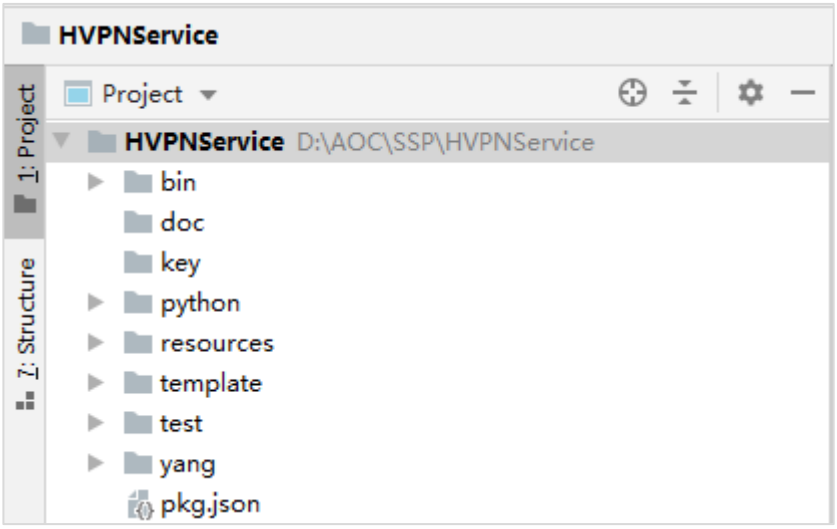
解压业务包到本地。



使用 Pycharm 打开此目录。



目录结构如下：



- bin: 可执行脚本存放位置，包含打包的工具及脚本。
- key: private 私钥存放位置。
- python: Python 代码存放位置，包含用来实现业务回调逻辑的 python 脚本。
- template: Jinja2 模板文件存放位置，包含用来实现完成映射后生成 NETCONF 报文所需的模板。
- test: 单元测试代码存放位置，包含用来对软件包功能进行本地单元测试的脚本。
- yang: 设备 YANG 模块。每个模块对应设备上的一个功能。它们共同组成设备 YANG 模型。
- pkg.json: 包配置文件，用来设置当前软件包的基本属性和回调钩子。

2.4.3 编写业务 YANG 模型

自定义的 VPN 网络业务由业务 YANG 模型承载。业务 YANG 模型描述了业务的相关参数。

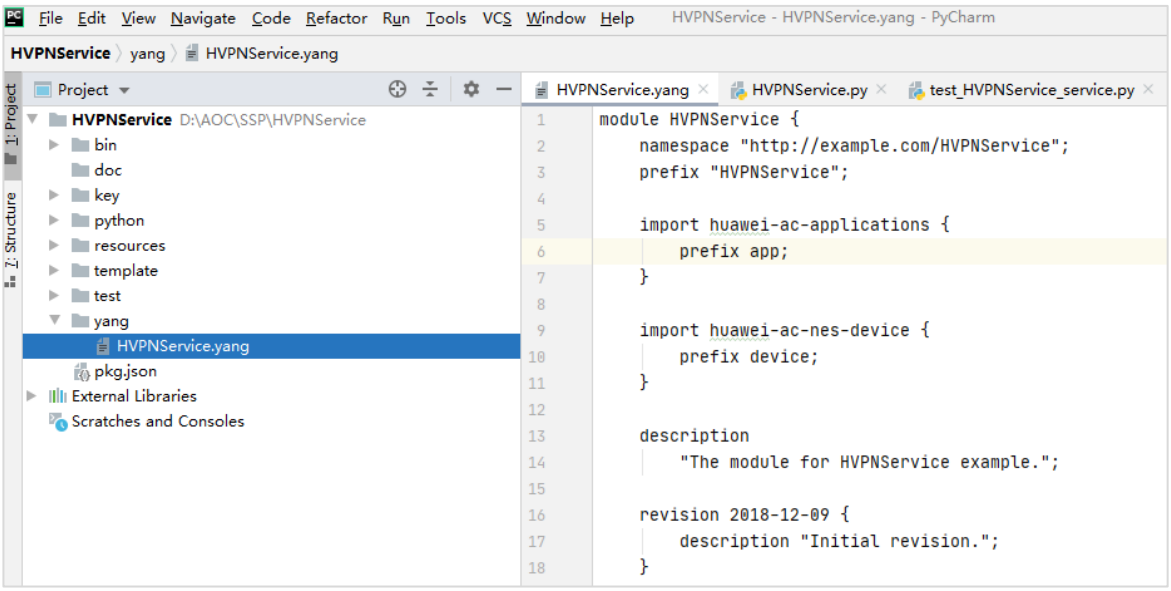
本小结在“yang”>“HVPNService.yang”文件中构建业务 YANG 模型。

```
module HVPNService {
    namespace "http://example.com/HVPNService";
    prefix "HVPNService";

    import huawei-ac-applications {
        prefix app;
    }

    import huawei-ac-nes-device {
        prefix device;
    }

    description
        "The module for HVPNService example.";

    revision 2018-12-09 {
        description "Initial revision.";
    }
```

```
    augment "/app:applications"{
        list hvpnService {
            app:application-definition "HVPNService";
            key "instanceName";
            leaf instanceName {
                type string;
            }
            list deviceList {
                key "deviceName";
                leaf deviceName {
                    type leafref {
                        path "/device:nes/device:ne/device:operate-name";
                    }
                    mandatory true;
                }
            }
            leaf name {
                type string;
            }
            leaf des {
                type string;
            }
            leaf mtu {
                type int32 {
                    range "46..9600";
                }
            }

            leaf vpn_instance_name {
                type string;
            }
        }
    }
}
```

本段代码读者可直接调用，不需要修改。

在 Pycharm 中呈现如图：



本例中定义业务 YANG 模块名为“HVPNService”，引入 huawei-ac-application 和 huawei-ac-nes-device 两个模块，将当前“HVPNService”模块扩展到 app 模块的/app.applications 下。

模块包含一个列表节点（List）名为“hvpnService”。列表定义了 “instanceName”、“deviceName”、“name”、“des”、“mtu”、“vpn_instance_name”等参数，用于用户在调用此业务时提交数据。

详细 YANG 原理请参考 HCIP-Datacom-Network Automation Developer 教材《NCE 业务开放可编程》和 RFC7950。

2.4.4 编写 Python 映射代码

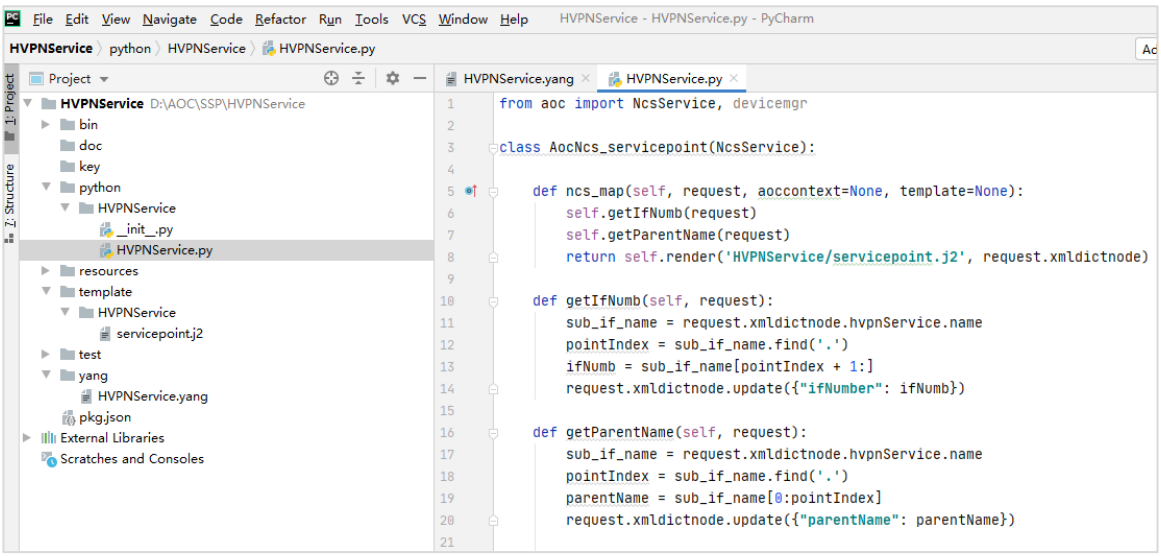
Python 映射代码描述了如何将用户提交的数据填充到模板，映射到网元数据结构中。

例如此处的 ifNumber 和 parentName 是从子接口参数中提取，然后更新到变量中提供给南向模板映射使用。

```
from aoc import NcsService, devicemgr

class AocNcs_servicepoint(NcsService):

    def ncs_map(self, request, aoccontext=None, template=None):
        self.getIfNumb(request)
        self.getParentName(request)
        return self.render('HVPNService/servicepoint.j2', request.xmldictnode)

    def getIfNumb(self, request):
        sub_if_name = request.xmldictnode.hvpnService.name
        pointIndex = sub_if_name.find('.')
        ifNumb = sub_if_name[pointIndex + 1:]
        request.xmldictnode.update({"ifNumber": ifNumb})

    def getParentName(self, request):
        sub_if_name = request.xmldictnode.hvpnService.name
        pointIndex = sub_if_name.find('.')
        parentName = sub_if_name[0:pointIndex]
```

```
        request.xmldictnode.update({"parentName": parentName})
```

在 Pycharm 中呈现如图：



代码解析：

```
from aoc import NcsService, devicemgr
```

“from aoc import NcsService”：引入需要使用的头文件，网元驱动包模板父类。

```
def ncs_map(self, request, aoccontext=None, template=None):
        self.getIfNumb(request)
        self.getParentName(request)
        return self.render('HVPNService/servicepoint.j2', request.xmldictnode)
```

“ncs_map”覆写父类中方法，编写获取设备创建接口需要的 ifNumb，ParentName 参数。

```
def getIfNumb(self, request):
        sub_if_name = request.xmldictnode.createInterfaces.name
        pointIndex = sub_if_name.find('.')
        ifNumb = sub_if_name[pointIndex + 1:]
        request.xmldictnode.update({"ifNumber": ifNumb})
```

定义函数 getIfNumb（self，request），用于从创建的子接口中获取出 ifNumb（本例中 GigabitEthernet3/0/0.20，取出 200）并更新到参数字典中，函数的一个参数 request 为北向入参。

```
def getParentName(self, request):
        sub_if_name = request.xmldictnode.createInterfaces.name
        pointIndex = sub_if_name.find('.')
        parentName = sub_if_name[0:pointIndex]
        request.xmldictnode.update({"parentName": parentName})
```

定义函数 getParentName（self，request），用于从创建的子接口中获取出 ParentName（本例中 GigabitEthernet3/0/0.20，取出 GigabitEthernet3/0/0）并更新到字典中，函数的第一个参数 request 为北向入参。

本小结简单介绍业务包 API 的使用，更多详细 API 介绍请参考《数通网络开放可编程开发指南》“API 参考”章节。

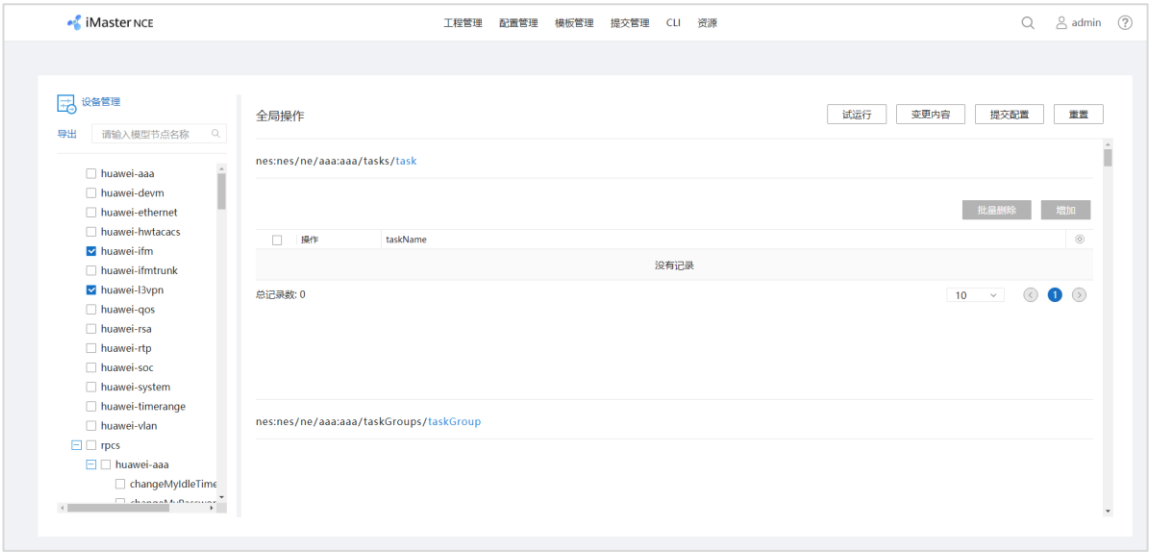
2.4.5 导出南向模板

南向模板用于和设备建立连接，和设备的类型和型号密切相关。

在主菜单中选择“配置管理” > “网元配置” > “设备管理”。单击操作栏中的编辑图标进入当前设备的配置信息界面。

因为本实验的网络业务的南向配置需要调用 huawei-ifm 和 huawei-l3vpn 两个设备 YANG 文件，用于编排子接口和 L3VPN 业务。

在搜索框中输入模块名（ifm 和 l3vpn），勾选 huawei-ifm 和 huawei-13vpn，单击导出按钮。



在弹出的导出页面，选择导出方式，勾选合并模板，使用“带数据不带结构”方式（携带基本信息），导出模板。

导出文件如下：

解压为.j2 格式文件，使用编辑器打开。此时文件中包含设备相关接口所有数据信息。

```
<ifm xmlns="http://www.huawei.com/netconf/vrp/huawei-ifm">
  <interfaces>
    <interface>
      <ifName>Virtual-Template0</ifName>
      <ifNumber>0</ifNumber>
      <statMode>baseInterface</statMode>
      <ifPhyType>Virtual-Template</ifPhyType>
      <ifControlFlap>
        <ifCtrlFlapEnbl>false</ifCtrlFlapEnbl>
      </ifControlFlap>
      <ifMtu>1492</ifMtu>
      <l2SubIfFlag>false</l2SubIfFlag>
      <ifTrapEnable>true</ifTrapEnable>
      <ifAdminStatus>up</ifAdminStatus>
      <ifClass>mainInterface</ifClass>
      <ipv4Config>
        <addrCfgType>config</addrCfgType>
      </ipv4Config>
      <ifStatiEnable>true</ifStatiEnable>
      <ipv6Config>
        <autoLinkLocal>false</autoLinkLocal>
        <enableFlag>false</enableFlag>
        <spreadMtu6Flag>false</spreadMtu6Flag>
      </ipv6Config>
      <ifLinkProtocol>ppp</ifLinkProtocol>
      <spreadMtuFlag>false</spreadMtuFlag>
      <ifRouterType>PtoP</ifRouterType>
```

根据“实验目标”需求，保留相关配置参数如下。

参数名	参数值
ifName	GigabitEthernet3/0/0.20
ifClass	sub-interface

ifParentIfName	GigabitEthernet3/0/0
ifNumber	20
IfDescr	For_Test_AOC
ifMtu	200
vrfName	Test_AOC

```
<ifm xmlns="http://www.huawei.com/netconf/vrp/huawei-ifm">
      <interface>
        <ifName>GigabitEthernet3/0/0.20</ifName>
        <ifNumber>20</ifNumber>
        <ifClass>subInterface</ifClass>
        <ifPhyType>GigabitEthernet</ifPhyType>
        <ifMtu>200</ifMtu>
        <ifDescr>For_Test_AOC</ifDescr>
        <ifParentIfName>GigabitEthernet3/0/0</ifParentIfName>
      </interface>
    </interfaces>
</ifm>

<l3vpn xmlns="http://www.huawei.com/netconf/vrp/huawei-l3vpn">
  <l3vpncomm>
    <l3vpnInstances>
      <l3vpnInstance>
        <vrfName>Test_AOC</vrfName>
        <l3vpnIfs>
          <l3vpnIf>
            <ifName>GigabitEthernet3/0/0.20</ifName>
          <l3vpnIf>
        </l3vpnIfs>
      </l3vpnInstance>
    </l3vpnInstances>
  </l3vpncomm>
</l3vpn>
```

2.4.6 编写南向模板

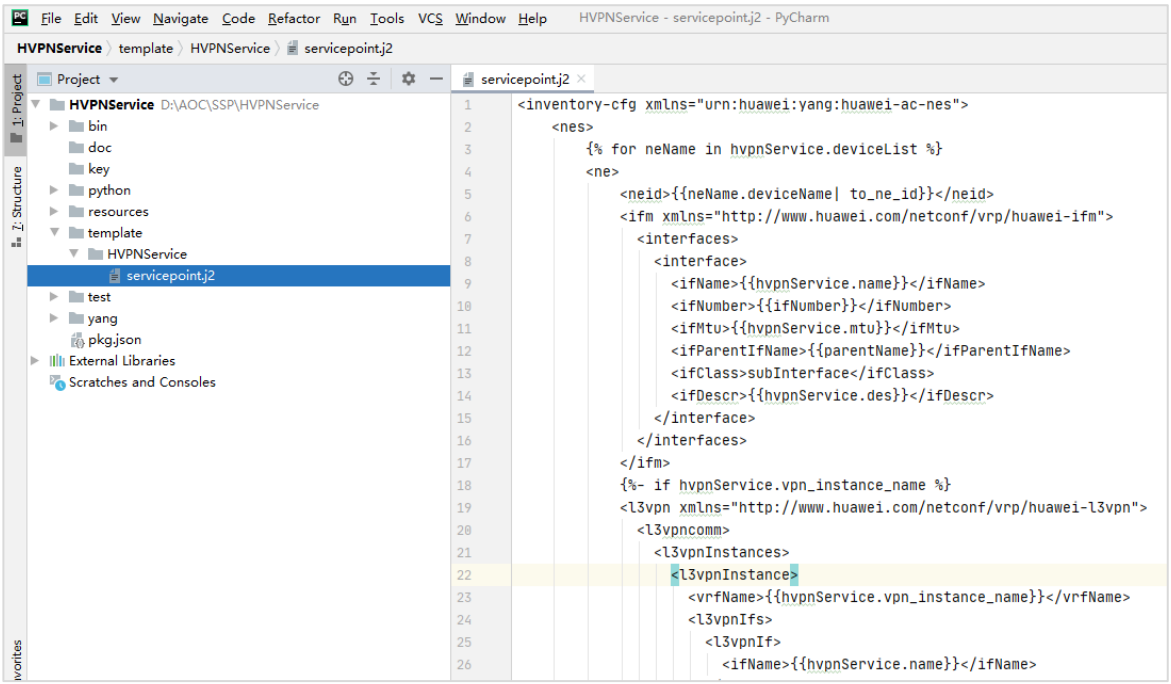
根据 ifm 和 l3vpn 的配置制作 jinja 模板，并将编写好的模板放入模板文件中（HVPNService>template>HVPNService>servicepoint.j2）。

```
<inventory-cfg xmlns="urn:huawei:yang:huawei-ac-nes">
    <nes>
        {% for neName in hvpnService.deviceList %}
        <ne>
            <neid>{{neName.deviceName| to_ne_id}}</neid>
            <ifm xmlns="http://www.huawei.com/netconf/vrp/huawei-ifm">
              <interfaces>
                <interface>
                  <ifName>{{hvpnService.name}}</ifName>
                  <ifNumber>{{ifNumber}}</ifNumber>
                  <ifMtu>{{hvpnService.mtu}}</ifMtu>
                  <ifParentIfName>{{parentName}}</ifParentIfName>
```

```
                    <ifClass>subInterface</ifClass>
                    <ifDescr>{{hvpnService.des}}</ifDescr>
                  </interface>
                </interfaces>
              </ifm>
              {%- if hvpnService.vpn_instance_name %}
              <l3vpn xmlns="http://www.huawei.com/netconf/vrp/huawei-l3vpn">
                <l3vpncomm>
                  <l3vpnInstances>
                    <l3vpnInstance>
                      <vrfName>{{hvpnService.vpn_instance_name}}</vrfName>
                      <l3vpnIfs>
                        <l3vpnIf>
                          <ifName>{{hvpnService.name}}</ifName>
                        </l3vpnIf>
                      </l3vpnIfs>
                    </l3vpnInstance>
                  </l3vpnInstances>
                </l3vpncomm>
              </l3vpn>
              {%- endif %}
          </ne>
          {% endfor %}
      </nes>
</inventory-cfg>
```

本段代码读者可直接调用，不需要修改。

在 Pycharm 中显示如图：



南向模板代码使用 jinja 模板。模板中包含变量和表达式，这两者在模板使用的过程中会被转换为对应的值。它有以下常用的语法：

- {% ... %} 中包含控制结构（Controll Structures）：在本例中{% for neName in hvpnService.deviceList %}表示进入 for 循环，{% endfor %}表示结束循环。
- {{…}}中包含表达式（Expression），表达式可以是常量、变量、数学公式和逻辑语句等。
- {# ... #} 表示注释。

更多 jiajia 语法和基本原理请参考官网 https://jinja.palletsprojects.com/en/2.11.x/。

代码解析：

```
<neid>{{neName.deviceName| to_ne_id}}</neid>
```

“to_ne_id”为过滤器，可以把设备名转换为系统识别的设备 id，对外展示的都是设备 id，系统内部都是使用 neid 来进行操作的，外部不需要感知。

```
<ifm xmlns="http://www.huawei.com/netconf/vrp/huawei-ifm">
  <interfaces>
    <interface>
      <ifName>{{hvpnService.name}}</ifName>
      <ifNumber>{{ifNumber}}</ifNumber>
      <ifMtu>{{hvpnService.mtu}}</ifMtu>
      <ifParentIfName>{{parentName}}</ifParentIfName>
      <ifClass>subInterface</ifClass>
      <ifDescr>{{hvpnService.des}}</ifDescr>
    </interface>
  </interfaces>
</ifm>
{%- if hvpnService.vpn_instance_name %}
<l3vpn xmlns="http://www.huawei.com/netconf/vrp/huawei-l3vpn">
  <l3vpncomm>
    <l3vpnInstances>
      <l3vpnInstance>
        <vrfName>{{hvpnService.vpn_instance_name}}</vrfName>
        <l3vpnIfs>
          <l3vpnIf>
            <ifName>{{hvpnService.name}}</ifName>
          </l3vpnIf>
        </l3vpnIfs>
      </l3vpnInstance>
    </l3vpnInstances>
  </l3vpncomm>
</l3vpn>
{%- endif %}
```

此部分代码是调用设备开放出来的原生能力对应的配置模板。该模板基于导出的南向模板，用来配置 ifm 和 l3vpn 接口信息，如 ifName 接口名、ifMtu 配置接口 MTU 等参数。

{{…}}中定义了多个变量，例如{{hvpnService.name}}、{{hvpnService.vpn_instance_name}}、{{hvpnService.des}}、{{hvpnService.name}}、{{ifNumber}}和{{parentName}}等。注意此处的的命名需要与 2.4.3 业务 YANG 模型和 2.4.4 Python 映射代码中参数匹配。

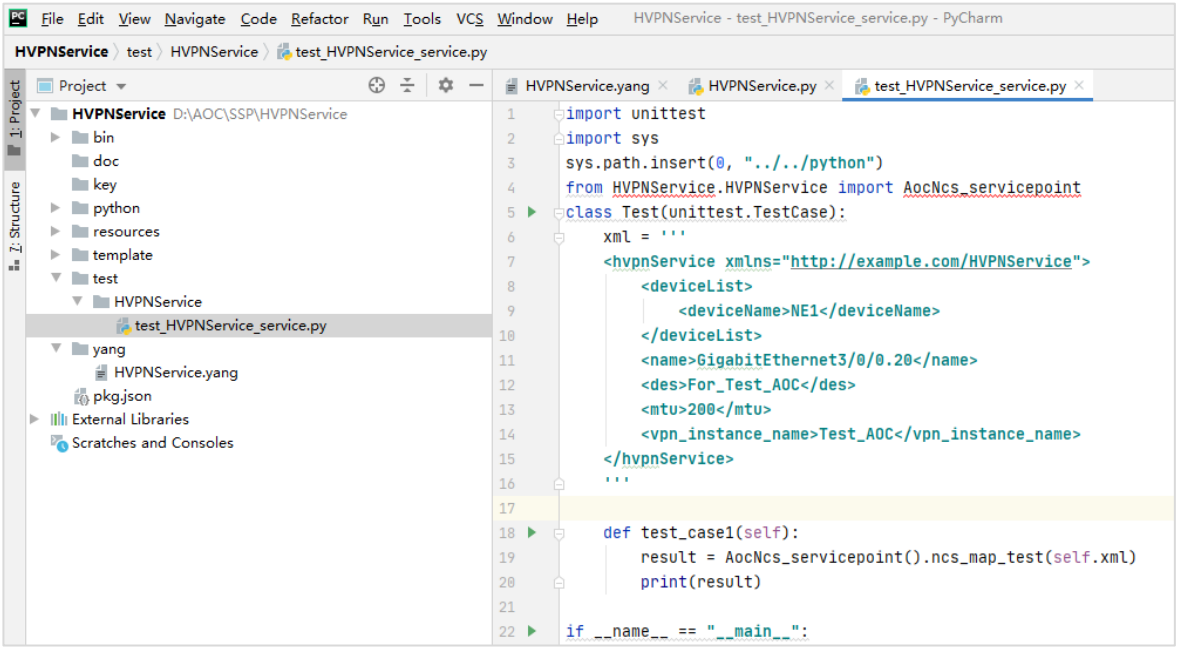
2.4.7 编写测试用例

编写测试用例，测试生成的配置报文是否正确。LLT（Low Level Test）测试代码框架在导出模板时也生成，只需要根据业务 YANG 模型，定义北向输入即可，LLT 中的北向输入以 XML 形式体现。

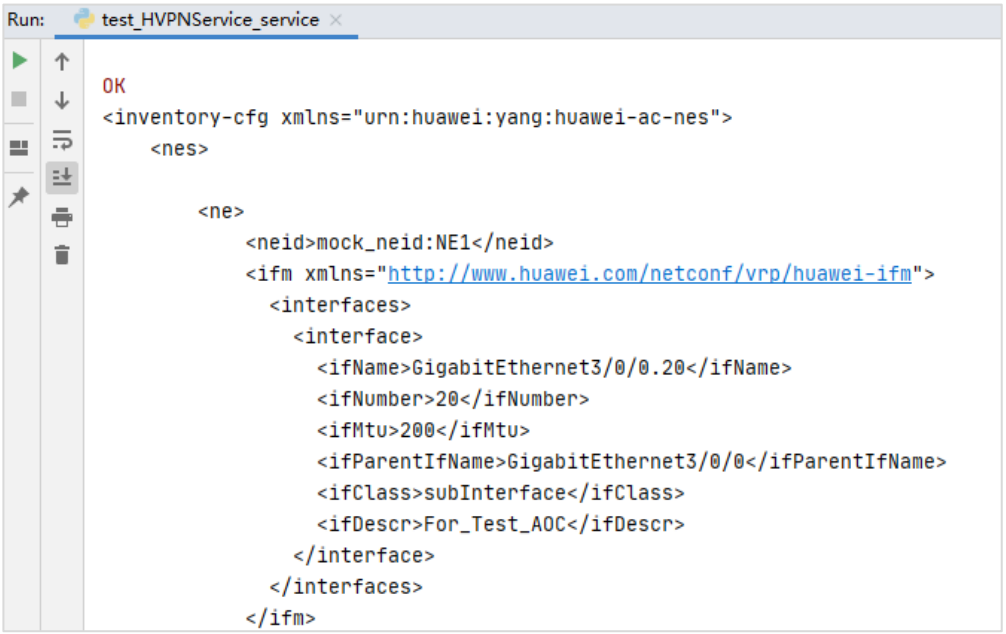
（test > HVPNService > test_HVPNService_service.py）

```
import unittest
import sys
sys.path.insert(0, "../../python")
from HVPNService.HVPNService import AocNcs_servicepoint
class Test(unittest.TestCase):
    xml = '''
    <hvpnService xmlns="http://example.com/HVPNService">
        <deviceList>
            <deviceName>NE1</deviceName>
        </deviceList>
        <name>GigabitEthernet3/0/0.20</name>
        <des>For_Test_AOC</des>
        <mtu>200</mtu>
        <vpn_instance_name>Test_AOC</vpn_instance_name>
    </hvpnService>
    '''

    def test_case1(self):
        result = AocNcs_servicepoint().ncs_map_test(self.xml)
        print(result)

if __name__ == "__main__":
    unittest.main()
```

本地的测试用例代码包含 XML。XML 参数值是根据北向 YANG 模型定义的北向结构，模拟北向用户参数输入，转换成南向下发到设备的具体报文



运行测试脚本，可看到生成的报文，检查输出是否正确。




```
<inventory-cfg xmlns="urn:huawei:yang:huawei-ac-nes">
    <nes>

        <ne>
            <neid>mock_neid:NE1</neid>
            <ifm xmlns="http://www.huawei.com/netconf/vrp/huawei-ifm">
              <interfaces>
                <interface>
                  <ifName>GigabitEthernet3/0/0.20</ifName>
                  <ifNumber>20</ifNumber>
                  <ifMtu>200</ifMtu>
                  <ifParentIfName>GigabitEthernet3/0/0</ifParentIfName>
                  <ifClass>subInterface</ifClass>
                  <ifDescr>For_Test_AOC</ifDescr>
                </interface>
              </interfaces>
            </ifm>
            <l3vpn xmlns="http://www.huawei.com/netconf/vrp/huawei-l3vpn">
              <l3vpncomm>
                <l3vpnInstances>
                  <l3vpnInstance>
                    <vrfName>Test_AOC</vrfName>
                    <l3vpnIfs>
                      <l3vpnIf>
                        <ifName>GigabitEthernet3/0/0.20</ifName>
                      </l3vpnIf>
                    </l3vpnIfs>
                  </l3vpnInstance>
                </l3vpnInstances>
              </l3vpncomm>
            </l3vpn>
        </ne>

    </nes>
```

```
</inventory-cfg>
```

2.4.8 配置秘钥

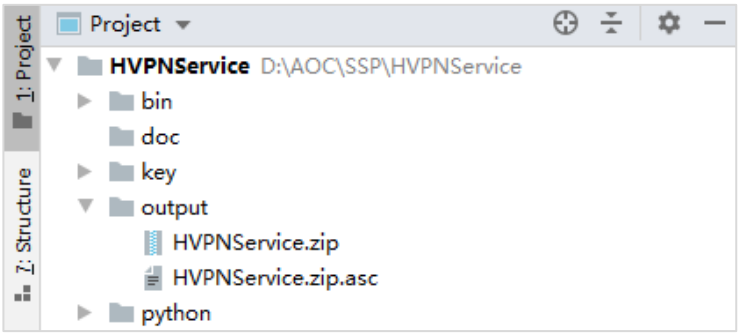
参考“网元驱动包实践”1.4.5“配置秘钥”章节（可复用），将 private.asc 文件复制到业务 SSP 包的 key 目录。

2.4.9 生成业务包（SSP）

使用 CMD 切换到 NE40E-X3 模板的 bin 目录，运行 makeFile.bat 文件，输入已设置的密钥密码，本例为“Huawei@123”。

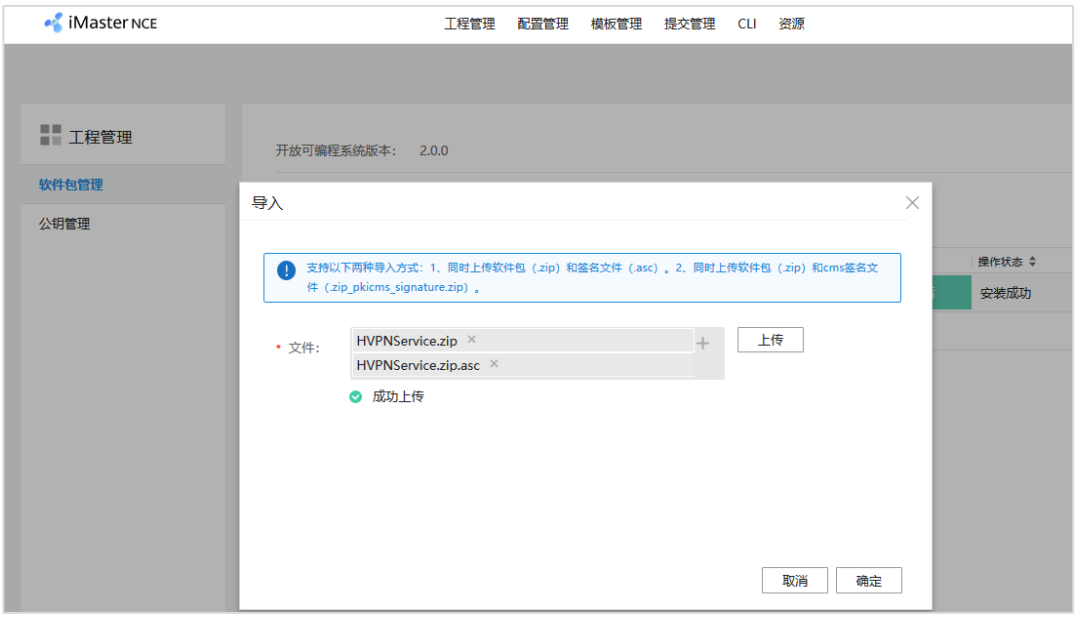
```
D:\AOC\SSP\HVPNService\bin>makeFile.bat
Please input password for private key:
…
2020-07-09 15:54:00,267 INFO [com.huawei.ncecommon.extended.pkg.mgr.tools.common.FileUtil] - [Sign]Generate signature file success.
2020-07-09 15:54:00,270 INFO [com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign] Sign: Execute success
```

回显信息包含“sucess”即成功业务包在 output 目录下



2.4.10 上传并激活业务包

在 iMaster NCE“工程管理”>“软件包管理”中导入网元驱动包。



点击“确定”，导入成功。

单击右侧“操作”中的“安装”按钮，激活驱动包。

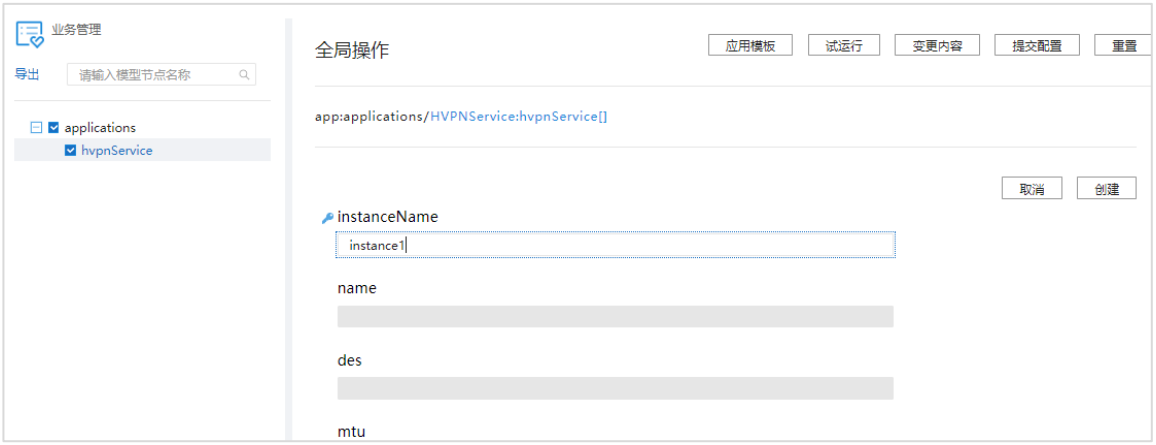
2.5 使用界面下发网络业务

2.5.1 下发网络业务

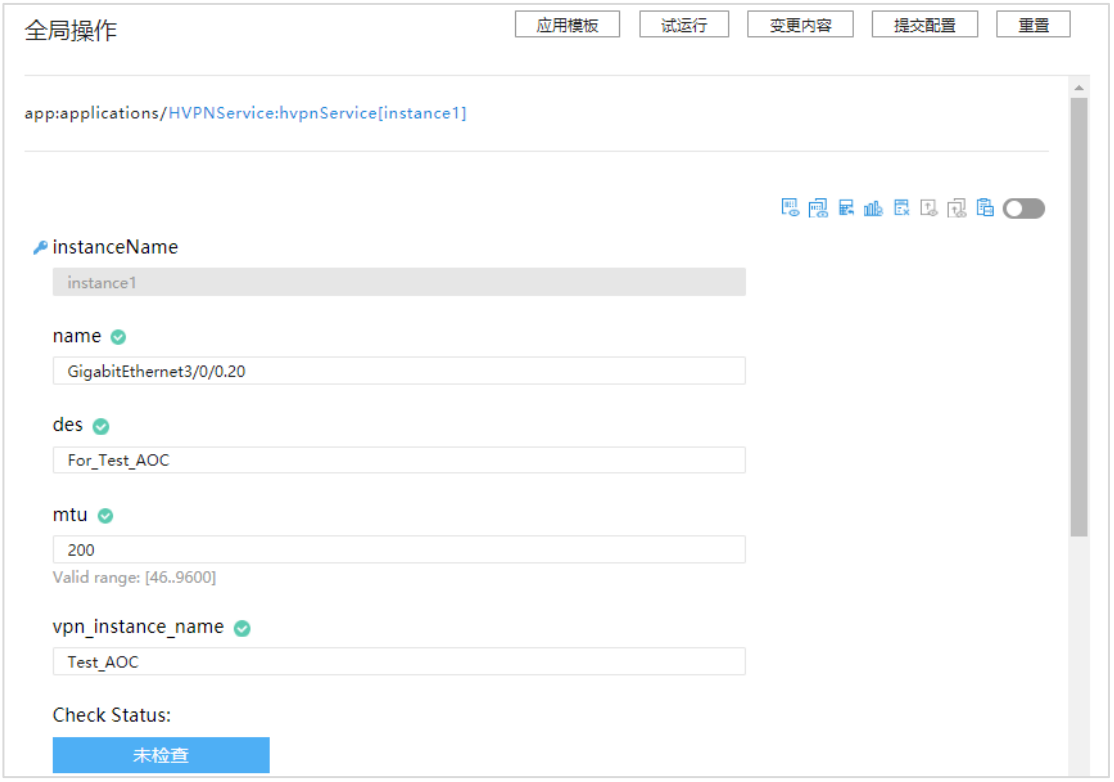
登录 iMaster NCE，进入“配置管理”>“业务管理”页面中。

选中业务模型“hvpnService”，点击“增加”。

首先填写实例名，“instanceName”中输入 instance1，点击“创建”。



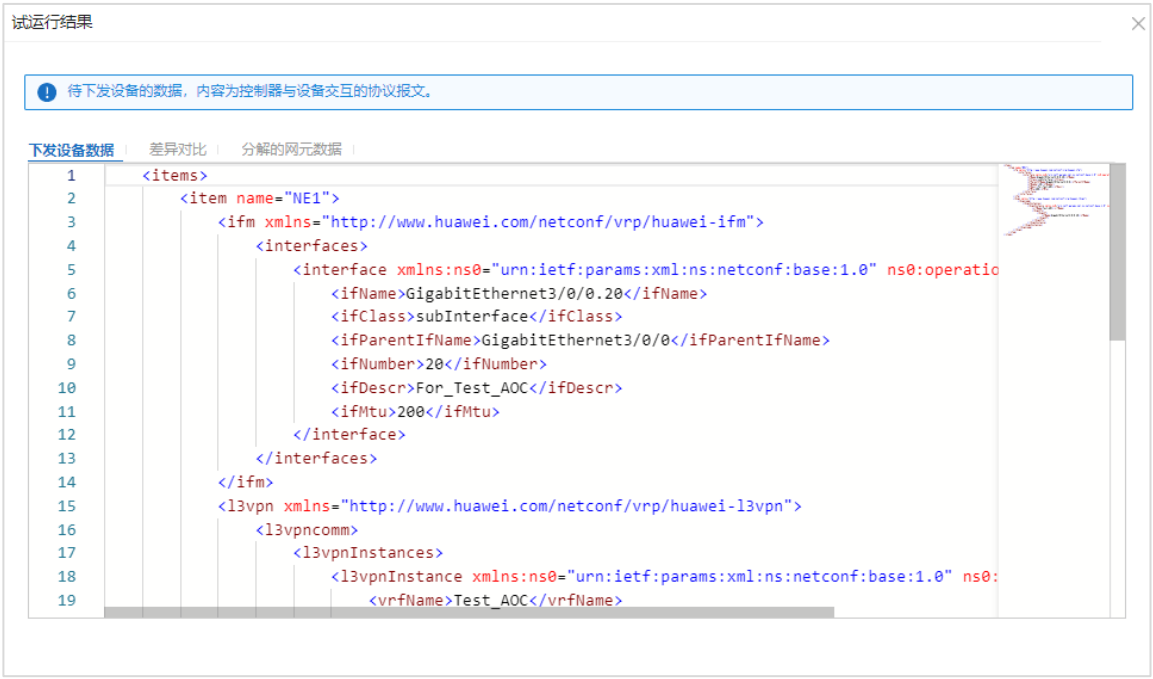
然后创建的子接口信息：“name”中输入接口 GigabitEthernet3/0/0.20，“des”输入 For_Test_AOC，“mtu”输入 200，“vpn_instance_name”输入 Test_AOC。



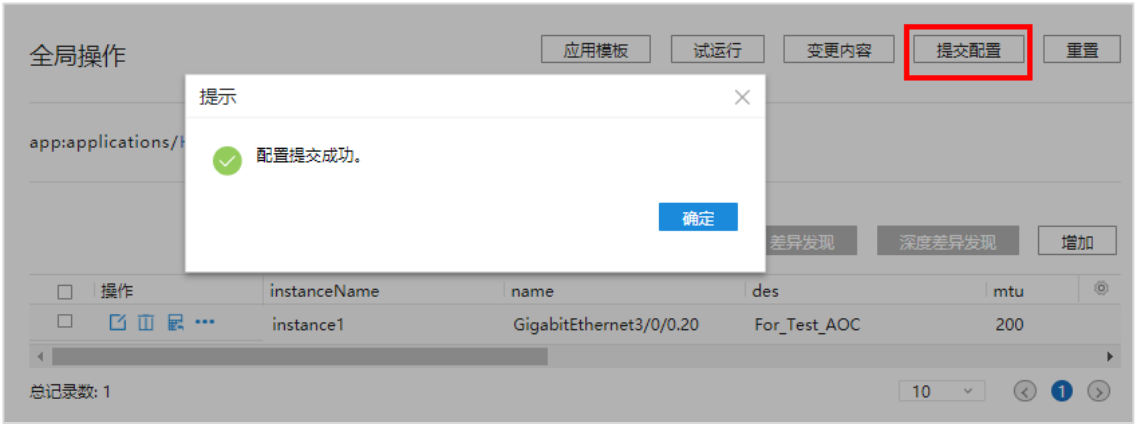
最后下拉推动条，在 deviceList 里点击“增加”选择添加要下发的设备。

选择“NE1”点击创建。

单击“试运行”，提前查看将要下发的配置报文，检查是否正确。

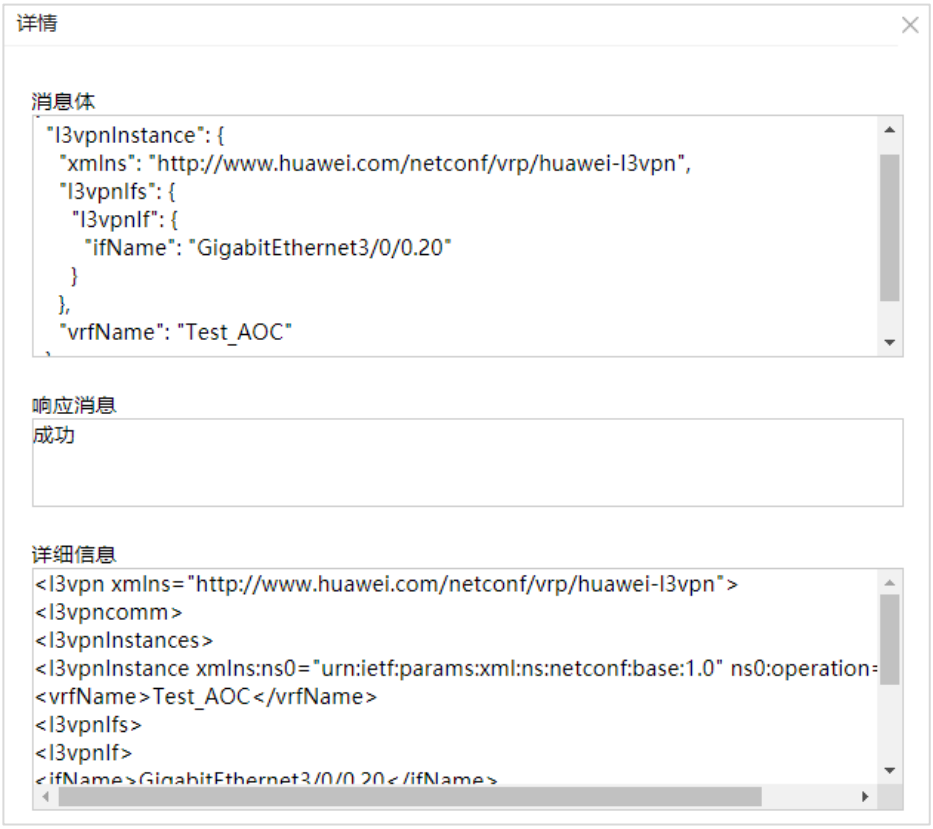


单击“提交配置”，将配置下发到设备上。



配置下发成功。可以在“配置管理”>“网元配置”>“设备配置历史”中，查看设备配置的变更情况。

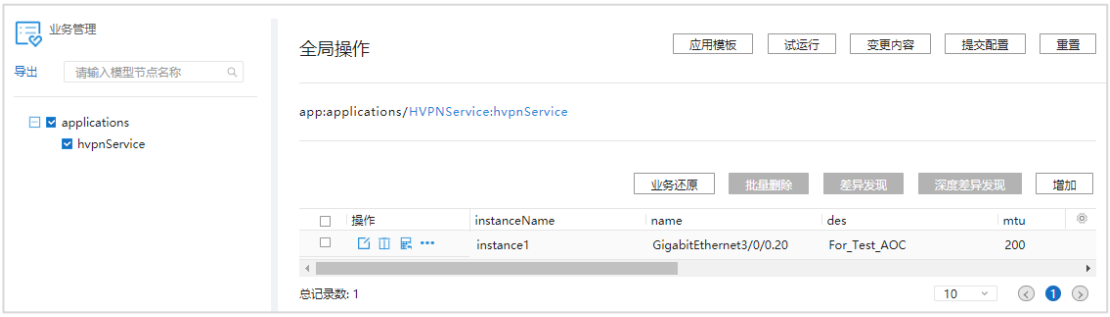
查看详情：



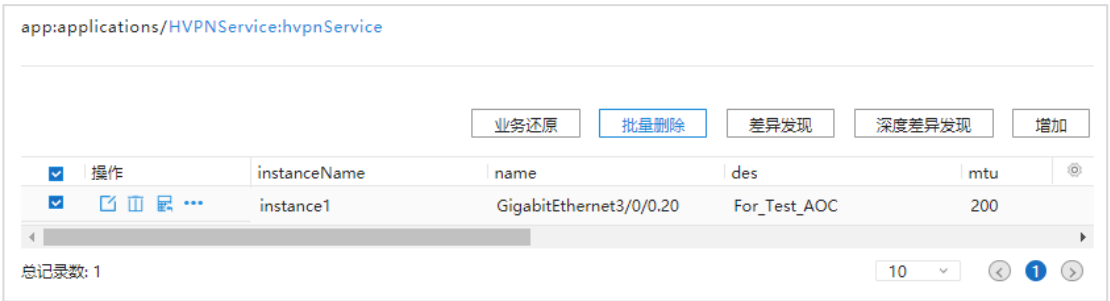
恭喜你成功完成自定义一个网络业务，并成功下发到设备。

2.5.2 删除网络业务

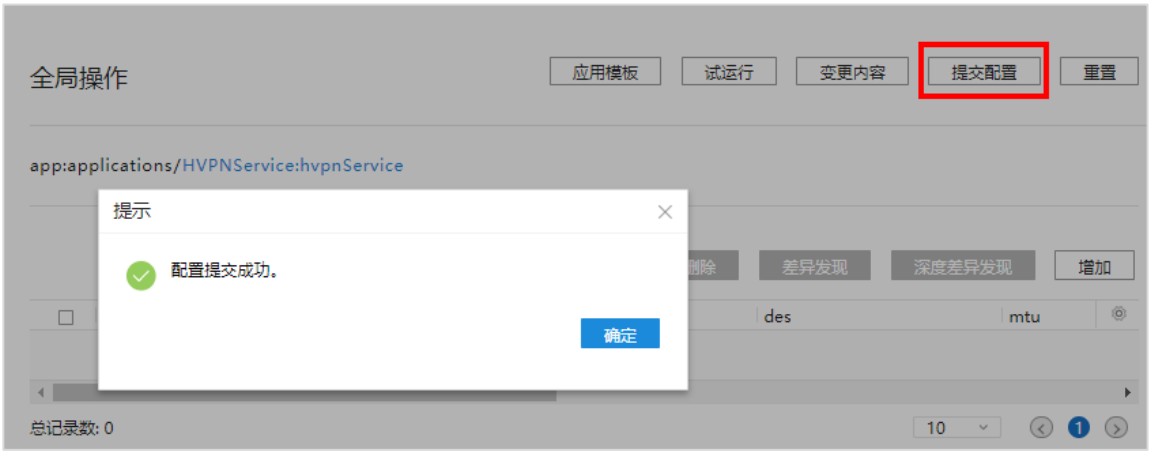
在“业务管理”页面查看当前网络业务。



选中需要删除的业务，点击“批量删除”。



点击“提交配置”。



查看设备配置变更情况：

业务被成功删除。

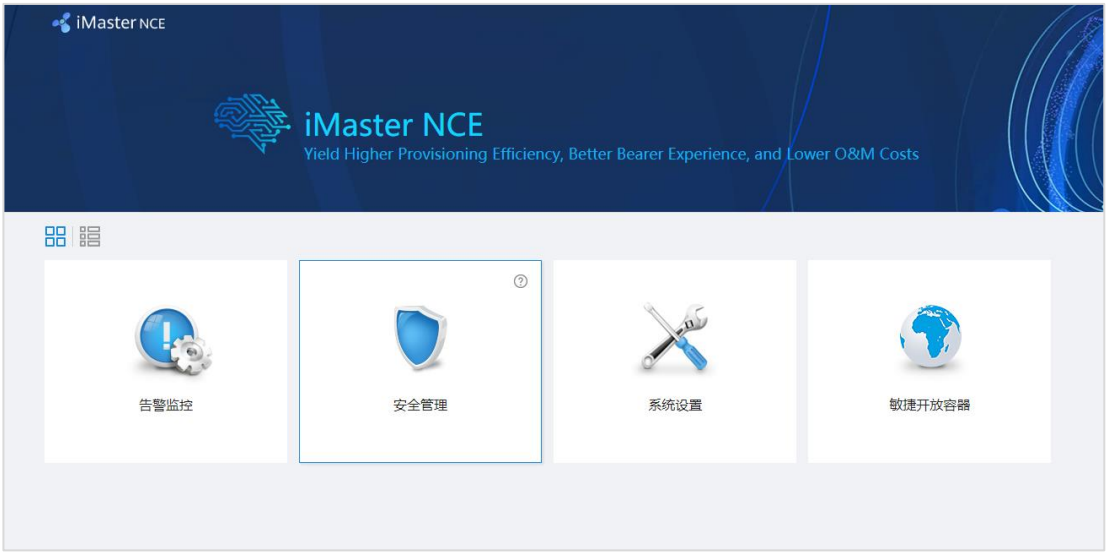
2.6 通过北向接口下发网络业务

iMaster NCE 可以基于定制业务包能力，提供北向开放开放能力。首先创建北向用户，然后使用北向账户实现“增删改查”。更多 iMaster NCE 北向能力请参考开发者社区“资源下载”《数通网络开放可编程北向 Rest+API 参考》和《数通网络开放可编程用户指南》“北向 API”章节。

本小节首先创建北向用户，然后通过北向接口下发业务相关配置：创建 VPN 业务实例并将子接口绑定到 VPN 上。

2.6.1 创建北向用户

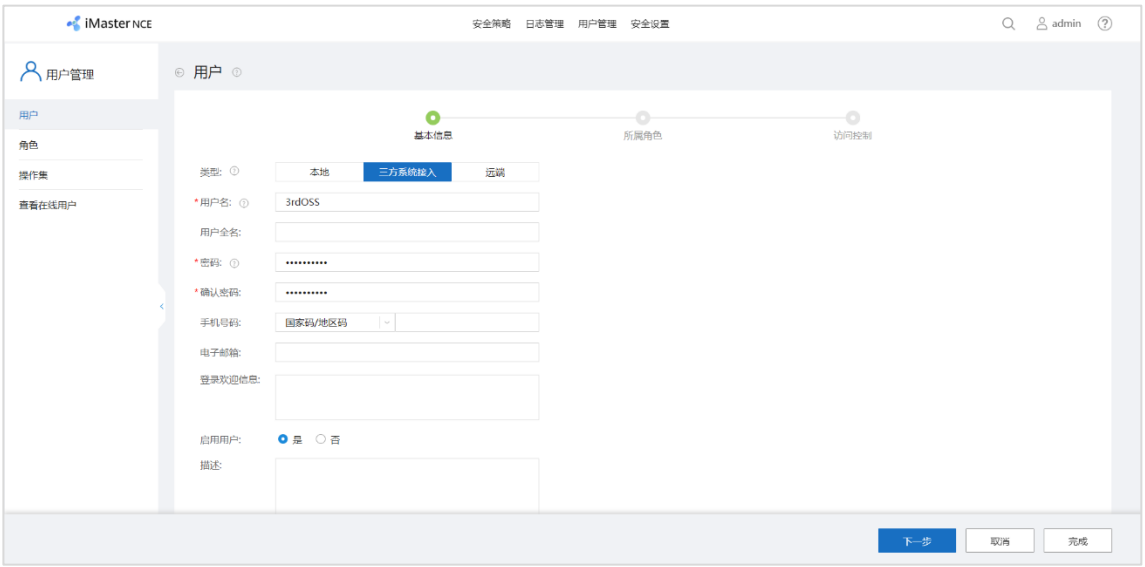
iMaster NCE 主界面单击“安全管理”。



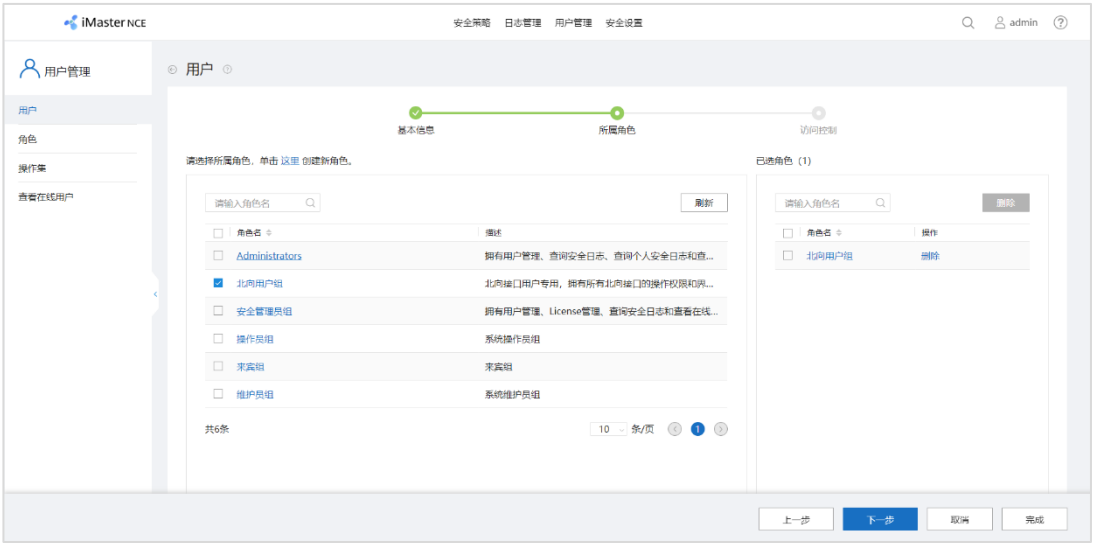
进入安全管理页面，单击“用户管理”按钮，单击“创建”。

进入创建用户页面，选择“三方系统接入”，添加用户名及密码，单击下一步 。

本例中创建用户名为“3rdOSS”，密码为“Huawei@123”。



勾选‘北向用户组’，单击“下一步”，后续按向导默认配置完成用户创建。



2.6.2 创建配置

1.获取 token

Method：PUT

```
https://{{northIP}}:26335/rest/plat/smapp/v1/oauth/token
```

请求头 Header

Key	Value
Content-Type	application/json

请求体

```
{
    "grantType":"password",
    "userName":"3rdOSS",
    "value":"Huawei@123"
}
```

返回体

```
{
      "accessSession": "x-
1cpceq1j0bdhft2niqiotg087uak6ps82ptfuqnw2pteqm84ob04o77u3vo4rtlg6mpdle2ktgo9dfml9cmrbzljs5u
k2k1f9ho7rw2pen6kqk49obdcamc8jzvs6llj",
      "roaRand": "48fa2a40d3250e6beb6a16cd806034574cf4fd9969a5f096",
      "expires": 1800,
      "additionalInfo": null
}
```

获取 token，下面所有发送的报文头中都需要带上 token。

2.申请事务

Method：POST

```
https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:create
```

请求头 Header，申请事务 id，请求头中包含步骤三中生成的 token

Key	Value
Content-Type	application/json
Accept	application/json
Cookie	accessSession={{accessSession}}

返回体中返回事务 id

```
{
    "huawei-ac-restconf-transactions:output": {
        "trans-id": "a4ce2a02-5e6f-41af-8824-852ba2155c2b"
    }
}
```

对业务进行编辑前，需要申请事务 ID。

3.编辑下发的配置（2.2.2 实验目标）

Method：POST

```
https://{{northIP}}:26335/restconf/v1/data/huawei-ac-applications:applications
```

请求头 Header 中携带步骤 1 生成的 token 和步骤 2 生成的事务 id。

Key	Value
restconf-transaction-id	{{transactionsID}}
Content-Type	application/json
Accept	application/json
Cookie	accessSession={{accessSession}}

请求体

```
{
    "HVPNService:hvpnService": [
        {
            "instanceName" : "instance1",
            "name" : "GigabitEthernet3/0/0.20",
            "des" : "For_Test_AOC",
            "mtu" : 200,
            "vpn_instance_name":"Test_AOC",
            "deviceList" : [
                {"deviceName" : "NE1"}
              ]
        }
    ]
}
```

4.试运行

Method：POST

```
https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:dry-run
```

请求头 Header

Key	Value
Content-Type	application/json
Accept	application/json
Cookie	accessSession={{accessSession}}

请求体

```
{
    "huawei-ac-restconf-transactions:input":{
        "trans-id":{{transactionsID}}
    }
}
```

返回体

```
{
    "huawei-ac-restconf-transactions:output": {
        "result": true
    }
}
```

编辑完业务后，只进行试运行，生成配置但是不下发到设备。

5.配置差异预览

Method：POST

```
https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:diff
```

请求头 Header

Key	Value
Content-Type	application/json
Accept	application/json
Cookie	accessSession={{accessSession}}

请求体

```
{
    "huawei-ac-restconf-transactions:input":{
        "trans-id": {{transactionsID}}
    }
}
```

返回体

```
{
    "huawei-ac-restconf-transactions:output": {
        "service-diff-infos": [
            {
                "feature": "/(urn:huawei:yang:huawei-ac-applications?revision=2018-07-
06)applications/AugmentationIdentifier{childNames=[(http://example.com/HVPNService?revision=2018-
```

```
12-09)hvpnService]}/(http://example.com/HVPNService?revision=2018-12-
09)hvpnService/hvpnService[{(http://example.com/HVPNService?revision=2018-12-
09)instanceName=instance1}]",
                    "diff-info": "{\"HVPNService:hvpnService\": [{\"left\": null, \"right\":
{\"[instanceName=instance1]\": {\"instanceName\": \"instance1\", \"name\":
\"GigabitEthernet1/0/8.201\", \"des\": \"testAOC\", \"mtu\": 50, \"vpn_instance_name\": \"testAOC202\",
\"deviceList\": [{\"[deviceName=NE1]\": {\"deviceName\": \"NE1\"}}]}}}]}"
                }
            ]
        }
}
```

编辑完配置后，预览当前事务的修改内容，返回网络层数据或网元层数据的差异。

6.查看试运行后的网元配置

Method：POST

```
https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:dry-run-query
```

请求头 Header

Key	Value
Content-Type	application/json
Accept	application/json
Cookie	accessSession={{accessSession}}

请求体

```
{
    "huawei-ac-restconf-transactions:input":{
        "trans-id": {{transactionsID}}
    }
}
```

返回体

```
{
    "huawei-ac-restconf-transactions:output": {
        "native": {
            "dry-run-ne-native-confs-item": [
                {
                    "ne-id": "0540f725-82b0-11ea-802c-f64ba7e1c586",
                    "ne-name": "NE1"
                }
            ]
        },
        "diff": {
            "dry-run-ne-diff-confs-item": [
                {
                    "ne-id": "0540f725-82b0-11ea-802c-f64ba7e1c586",
                    "ne-name": "NE1"
                }
            ]
        },
        "mapconf": {
```

```
                "service_map_confs": [
                    {
                        "source": "/huawei-ac-
applications:applications/HVPNService:hvpnService/instance1",
                        "dry-run-map-confs-item": [
                            {
                                "ne-id": "0540f725-82b0-11ea-802c-f64ba7e1c586",
                                "ne-name": "NE1",
                                "ne-cfg": [
                                    "<interface xmlns=\"http://www.huawei.com/netconf/vrp/huawei-
ifm\">\n<ifName>GigabitEthernet3/0/0.20</ifName>\n<ifNumber>20</ifNumber>\n<ifClass>subInterfac
e</ifClass>\n<ifMtu>200</ifMtu>\n<ifDescr>For_Test_AOC</ifDescr>\n<ifParentIfName>GigabitEthernet
3/0/0</ifParentIfName>\n</interface>\n",
                                    "<l3vpnInstance
xmlns=\"http://www.huawei.com/netconf/vrp/huawei-
l3vpn\">\n<vrfName>Test_AOC</vrfName>\n<l3vpnIfs>\n<l3vpnIf>\n<ifName>GigabitEthernet3/0/0.20<
/ifName>\n</l3vpnIf>\n</l3vpnIfs>\n</l3vpnInstance>\n"
                                ]
                            }
                        ]
                    }
                ]
            }
        }
}
```


查看试运行后生成的配置和差异信息。

7.提交事务

Method：POST

```
https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:commit
```

请求头 Header

Key	Value
Content-Type	application/json
Accept	application/json
Cookie	accessSession={{accessSession}}

请求体

```
{
    "huawei-ac-restconf-transactions:input":{
        "force":false,
        "trans-id": {{transactionsID}},
        "no-network":true
    }
}
```

返回体

```
{
    "huawei-ac-restconf-transactions:output": {
        "result": true,
```

```
            "reason": ""
        }
}
```

业务编辑结束后，将编辑的数据提交到控制器上，使其正式生效。

2.6.3 查看配置

1.获取 token

Method：PUT

```
https://{{northIP}}:26335/rest/plat/smapp/v1/oauth/token
```

请求头 Header

Key	Value
Content-Type	application/json
Accept	application/json

请求体

```
{
    "grantType":"password",
    "userName":"3rdOSS",
    "value":"Huawei@123"
}
```

返回体样例

```
{       "accessSession": "x-
1cpceq1j0bdhft2niqiotg087uak6ps82ptfuqnw2pteqm84ob04o77u3vo4rtlg6mpdle2ktgo9dfml9cmrbzljs5u
k2k1f9ho7rw2pen6kqk49obdcamc8jzvs6llj",
        "roaRand": "48fa2a40d3250e6beb6a16cd806034574cf4fd9969a5f096",
        "expires": 1800,
        "additionalInfo": null
}
```

返回报文中的 accessSession 就是 token，下面所有发送的报文头中都需要带上 token，进行鉴权。

2、查看配置

Method：GET

```
https://{{northIP}}:26335/restconf/v1/data/huawei-ac-
applications:applications/HVPNService:hvpnService/instance1
```

请求头 Header

Key	Value
Accept	application/json
Cookie	accessSession={{accessSession}}
Content-Type	application/json

返回体

```
{
    "HVPNService:hvpnService": [
        {
            "instanceName": "instance1",
            "name": "GigabitEthernet3/0/0.20",
            "des": "For_Test_AOC",
            "mtu": 200,
            "vpn_instance_name": "Test_AOC",
            "deviceList": [
                {
                    "deviceName": "NE1"
                }
            ]
        }
    ]
}
```

2.6.4 修改配置

1.获取 token

Method：PUT

```
https://{{northIP}}:26335/rest/plat/smapp/v1/oauth/token
```

请求头 Header

Key	Value
Content-Type	application/json
Accept	application/json

请求体

```
{
   "grantType":"password",
   "userName":"3rdOSS",
   "value":"Huawei@123"
}
```

返回体样例

```
{     "accessSession": "x-
1cpceq1j0bdhft2niqiotg087uak6ps82ptfuqnw2pteqm84ob04o77u3vo4rtlg6mpdle2ktgo9dfml9cmrbzljs5u
k2k1f9ho7rw2pen6kqk49obdcamc8jzvs6llj",
     "roaRand": "48fa2a40d3250e6beb6a16cd806034574cf4fd9969a5f096",
     "expires": 1800,
     "additionalInfo": null
}
```

返回报文中的 accessSession 就是 token，下面所有发送的报文头中都需要带上 token，进行鉴权。

2.申请事务

Method：POST

```
https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:create
```

请求头 Header，请求头中包含生成的 token。

Key	Value
Content-Type	application/json
Accept	application/json
Cookie	accessSession={{accessSession}}

返回体样例，其中 trans-id 就是新建的事务 id：

```
{
    "huawei-ac-restconf-transactions:output": {
        "trans-id": "a4ce2a02-5e6f-41af-8824-852ba2155c2b"
    }
}
```

3.修改配置（修改 MTU 为 50）

Method：PUT

```
https://{{northIP}}:26335/restconf/v1/data/huawei-ac-
applications:applications/CreateInterfaceService:createInterfaces/instance1
```

请求头 Header

Key	Value
restconf-transaction-id	{{transactionsID}}
Accept	application/json
Cookie	accessSession={{accessSession}}
Content-Type	application/json

请求体

```
{
    "HVPNService:hvpnService": [
        {
            "instanceName" : "instance1",
   "name" : "GigabitEthernet3/0/0.20",
   "des" : "For_Test_AOC",
   "mtu" : 50,
   "vpn_instance_name":"Test_AOC",
   "deviceList" : [
       {"deviceName" : "NE1"}
   ]
        }
    ]
}
```

4.试运行

Method：POST

```
https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:dry-run
```

请求头 Header

Key	Value
restconf-transaction-id	{{transactionsID}}
Accept	application/json
Cookie	accessSession={{accessSession}}
Content-Type	application/json

请求体

```
{
    "huawei-ac-restconf-transactions:input":{
        "trans-id":{{transactionsID}}
    }
}
```

返回体样例

```
{
    "huawei-ac-restconf-transactions:output": {
        "result": true
    }
}
```

5.配置差异预览

Method：POST

```
https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:diff
```

请求头 Header

Key	Value
Content-Type	application/json
Accept	application/json
Cookie	accessSession={{accessSession}}

请求体

```
{
    "huawei-ac-restconf-transactions:input":{
        "trans-id": {{transactionsID}}
    }
}
```

返回体

```
{
    "huawei-ac-restconf-transactions:output": {
        "service-diff-infos": [
            {
```

```
                    "feature": "/(urn:huawei:yang:huawei-ac-applications?revision=2018-07-
06)applications/AugmentationIdentifier{childNames=[(http://example.com/HVPNService?revision=2018-
12-09)hvpnService]}/(http://example.com/HVPNService?revision=2018-12-
09)hvpnService/hvpnService[{(http://example.com/HVPNService?revision=2018-12-
09)instanceName=instance1}]",
                    "diff-info": "{\"hvpnService\": [{\"[instanceName=instance1]\": {\"instanceName\":
\"instance1\", \"mtu\": {\"left\": 200, \"right\": 50}}}]}"
                }
            ]
        }
}
```

编辑完配置后，预览当前事务的修改内容，返回网络层数据或网元层数据的差异。

6.查看试运行后的网元配置

Method：POST

```
https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:dry-run-query
```

请求头 Header

Key	Value
Content-Type	application/json
Accept	application/json
Cookie	accessSession={{accessSession}}

请求体

```
{
    "huawei-ac-restconf-transactions:input":{
        "trans-id": {{transactionsID}}
    }
}
```

返回体

```
{
    "huawei-ac-restconf-transactions:output": {
        "native": {
            "dry-run-ne-native-confs-item": [
                {
                    "ne-id": "0540f725-82b0-11ea-802c-f64ba7e1c586",
                    "ne-name": "NE1",
                    "ne-cfg": [
                        "<ifm xmlns=\"http://www.huawei.com/netconf/vrp/huawei-
ifm\">\n<interfaces>\n<interface xmlns:ns0=\"urn:ietf:params:xml:ns:netconf:base:1.0\"
ns0:operation=\"merge\">\n<ifName>GigabitEthernet3/0/0.20</ifName>\n<ifMtu>50</ifMtu>\n</interfa
ce>\n</interfaces>\n</ifm>\n"
                    ]
                }
            ]
        },
        "diff": {
            "dry-run-ne-diff-confs-item": [
                {
```

```
                        "ne-id": "0540f725-82b0-11ea-802c-f64ba7e1c586",
                        "ne-name": "NE1",
                        "ne-cfg-diffs": [
                            {
                                "feature": "/(http://www.huawei.com/netconf/vrp/huawei-
ifm?revision=2018-06-11)ifm",
                                "diff-info": "{\"ifm\": {\"interfaces\": {\"interface\":
[{\"[ifName=GigabitEthernet3/0/0.20]\": {\"ifName\": \"GigabitEthernet3/0/0.20\", \"ifMtu\": {\"left\":
200, \"right\": 50}}}]}}}"
                            }
                        ]
                    }
                ]
            },
            "mapconf": {
                "service_map_confs": [
                    {
                        "source": "/huawei-ac-
applications:applications/HVPNService:hvpnService/instance1",
                        "dry-run-map-confs-item": [
                            {
                                "ne-id": "0540f725-82b0-11ea-802c-f64ba7e1c586",
                                "ne-name": "NE1",
                                "ne-cfg": [
                                    "<interface xmlns=\"http://www.huawei.com/netconf/vrp/huawei-
ifm\">\n<ifName>GigabitEthernet3/0/0.20</ifName>\n<ifNumber>20</ifNumber>\n<ifClass>subInterfac
e</ifClass>\n<ifMtu>50</ifMtu>\n<ifDescr>For_Test_AOC</ifDescr>\n<ifParentIfName>GigabitEthernet3
/0/0</ifParentIfName>\n</interface>\n",
                                    "<l3vpnInstance
xmlns=\"http://www.huawei.com/netconf/vrp/huawei-
l3vpn\">\n<vrfName>Test_AOC</vrfName>\n<l3vpnIfs>\n<l3vpnIf>\n<ifName>GigabitEthernet3/0/0.20<
/ifName>\n</l3vpnIf>\n</l3vpnIfs>\n</l3vpnInstance>\n"
                                ]
                            }
                        ]
                    }
                ]
            }
        }
}
```


查看试运行后生成的配置和差异信息。

6.提交配置

Method：POST

```
https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:commit
```

请求头 Header

Key	Value
Content-Type	application/json
Accept	application/json

Cookie	accessSession={{accessSession}}

请求体

```
{
    "huawei-ac-restconf-transactions:input":{
        "force":false,
        "trans-id": {{transactionsID}},
        "no-network":true
    }
}
```

返回体

```
{
    "huawei-ac-restconf-transactions:output": {
        "result": true,
        "reason": ""
    }
}
```

2.6.5 删除配置

1.获取 token

Method：PUT

```
https://{{northIP}}:26335/rest/plat/smapp/v1/oauth/token
```

请求头 Header

Key	Value
Content-Type	application/json

请求体

```
{
   "grantType":"password",
   "userName":"3rdOSS",
   "value":"Huawei@123"
}
```

返回体

```
{
    "accessSession": "x-
1cpceq1j0bdhft2niqiotg087uak6ps82ptfuqnw2pteqm84ob04o77u3vo4rtlg6mpdle2ktgo9dfml9cmrbzljs5u
k2k1f9ho7rw2pen6kqk49obdcamc8jzvs6llj",
    "roaRand": "48fa2a40d3250e6beb6a16cd806034574cf4fd9969a5f096",
    "expires": 1800,
    "additionalInfo": null
}
```

获取 token，下面所有发送的报文头中都需要带上 token。

2.申请事务

Method：POST

```
https://{{northIP}}:26335/restconf/operations/huawei-ac-restconf-transactions:create
```

请求头 Header 包含生成的 token。

Key	Value
Content-Type	application/json
Accept	application/json
Cookie	accessSession={{accessSession}}

返回体中返回事务 id

```
{
    "huawei-ac-restconf-transactions:output": {
        "trans-id": "a4ce2a02-5e6f-41af-8824-852ba2155c2b"
    }
}
```

对业务进行编辑前，需要申请事务 ID。

3.删除配置

Method：DELETE

```
https://{{northIP}}:26335/restconf/v1/data/huawei-ac-
applications:applications/HVPNService:hvpnService/instance1
```

请求头 Header

Key	Value
restconf-transaction-id	{{transactionsID}}
Accept	application/json
Cookie	accessSession={{accessSession}}
Content-Type	application/json

返回体：200

2.7 思考题

如果客户有自有的 OSS 系统，如何根据客户的需求开发业务模型，并完成对接?

思考题参考答案

《网元驱动包实践》参考答案：

1.设备升级后如果对应的 YANG 模型如果有变化，需要重新针对版本重新开发网元驱动包。

2.iMaster NCE 的设备添加支持批量导入。可以预先将要添加的设备填写到导入模板中，一次性的导入多台设备。

3.旧设备款型支持命令行，可以使用 SSH 进行对接 iMaster NCE 然后下发 CLI，具体操作可以参考版本发布的文档《数通网络开放可编程开发指南》。

4.iMaster NCE 会自动根据设备的 YANG 模型生成北向接口，供上层 OSS 系统对接使用。。

《业务包实践》参考答案：

1.需要梳理一个完整的网络业务需要下发的所有配置。

2.梳理这些配置所需的外部输入参数。由此基本确定了北向 OSS 系统对接需要的入参。

3.根据北向输入参数决定业务 YANG 模型。开放可编程会自动根据业务 YANG 模型生成北向接口，供上层 OSS 系统对接使用。